EDGEWOOD

RESEARCH, DEVELOPMENT & ENGINEERING CENTER

U.S. ARMY CHEMICAL AND BIOLOGICAL DEFENSE COMMAND

ERDEC-TR-128

REAL-TIME DATA COLLECTION PROGRAMS AND SOURCE CODE FOR A COMMERCIAL PASSIVE FTIR REMOTE SENSOR

Robert Kroutil

RESEARCH AND TECHNOLOGY DIRECTORATE

Michael Housky

MIDAC CORPORATION
Irvine, CA 92714

Gary S. Small

OHIO UNIVERSITY
Athens, OH 45701

August 1994

CUM SCIENTIA DEFENDIMUS

Approved for public release; distribution is unlimited.

Aberdeen Proving Ground, MD 21010-5423

Disclaimer

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
1994 August

3. REPORT TYPE AND DATES COVERED
Final, 92 Sep - 93 Jul

4. TITLE AND SUBTITLE
Real-Time Data Collection Programs and Source Code for a Commercial Passive FTIR Remote Sensor

5. FUNDING NUMBERS
PR-10162622A553
DARPA Project AO-8304

6. AUTHOR(S)
Kroutil, Robert (ERDEC);* Housky, Michael (Midac Corporation); and Small, Gary S. (Ohio University)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DIR, ERDEC, ATTN: SCBRD-RTM, APG, MD 21010-5423

Midac Corporation, Irvine, CA 92714

Ohio University, Athens, OH 45701

8. PERFORMING ORGANIZATION REPORT NUMBER
ERDEC-TR-128

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
*When this study was conducted, ERDEC was known as the U.S. Army Chemical Research, Development and Engineering Center, and the ERDEC author was assigned to the Research Directorate.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Computer programs for data collection and analysis of interferograms from a commercial Fourier Transform Infrared sensor were developed. Programs written in "C" language for an IBM PC using the DOS operating system allow one to collect and display data in a variety of formats useful for the environmental monitoring of vapor clouds. Software is described that enables the user to execute signal processing algorithms for the real-time analysis of interferograms. The programs collect data from the commercial interferometer and detect a vapor species using digital filters and pattern recognition methods. Source code and documentation describing all program functions are provided.

14. SUBJECT TERMS
Fourier Transform Infrared
Infrared background suppression
Passive infrared software
Infrared
FTIR

15. NUMBER OF PAGES
230

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Blank

PREFACE

The work described in this report was authorized under Project No. 10162622A553, CB Defense/General Investigation. This work was started in September 1992 and completed in July 1993.

The use of trade names or manufacturers' names in this report does not constitute an official endorsement of any commercial products. This report may not be cited for purposes of advertisement.

This report has been approved for release to the public. Registered users should request additional copies from the Defense Technical Information Center; unregistered users should direct such requests to the National Technical Information Service.

Acknowledgments

This work was supported by DARPA project A0-8304. The authors wish to acknowledge Dr. Roger Combs and Robert Knapp of the U.S. Army Edgewood Research, Development and Engineering Center (ERDEC),* Aberdeen Proving Ground, MD. Dr. Combs and Robert Knapp provided many extremely useful recommendations that greatly improved the functionality of the PC software implementation. The authors also wish to thank the Midac Corporation (Irvine, CA) for their support of this work.

Accesion For	
NTIS CRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

*When this study was conducted, ERDEC was known as the U.S. Army Chemical Research, Development and Engineering Center, and the ERDEC author was assigned to the Research Directorate.

Blank

CONTENTS

Blank

REAL-TIME DATA COLLECTION PROGRAMS AND SOURCE CODE FOR A COMMERCIAL PASSIVE FTIR REMOTE SENSOR

1. INTRODUCTION

Passive Fourier Transform Infrared (FTIR) remote infrared (IR) sensing is a method of increasing popularity for the detection and identification of gaseous pollutants. Low-cost, commercial FTIR remote sensors are available that can detect the spectral absorptions or emissions of a chemical vapor cloud, using an ambient temperature thermal background. Current data analysis computer programs require the collection of an ambient background reference spectrum for subtraction. Many conditions exist in the detection of ambient vapor clouds in which it is not possible to obtain an accurate passive open-path background emission spectrum. These conditions include the use of a passive FTIR sensor in a fence-line or an open-path smoke stack monitoring application.

A typical fence-line monitoring application is shown in Figure 1. This application detection limit is governed by four basic parameters, which are as follows:

- the contrasting temperature between the background and the vapor cloud
- the cloud path length
- the cloud concentration
- the atmospheric attenuation of the IR energy

In addition, the spectral features of a chemical vapor cloud can be seen as either emission or absorption depending on the relative temperature of the cloud relative to that temperature of the ambient background. Because of the large number of atmospheric and instrumental variables, a passive IR spectrum can be so complicated that the identification of a particular vapor species can be easily missed. Automated real-time pattern recognition software can assist the operator in discriminating between the complicated background spectra and the spectral features of a vapor cloud.

Recently, a generic series of signal processing techniques has been developed to analyze passive remote sensing interferograms.[1-5] These methods use a digital filter combined with pattern recognition that can discriminate a particular chemical species versus all other interferents and spectral background features. A digital filter that is used is called a "matrix" filter. This filter is somewhat analogous to the background reference subtraction in that it removes the unwanted spectral background features before application of a pattern recognition procedure.

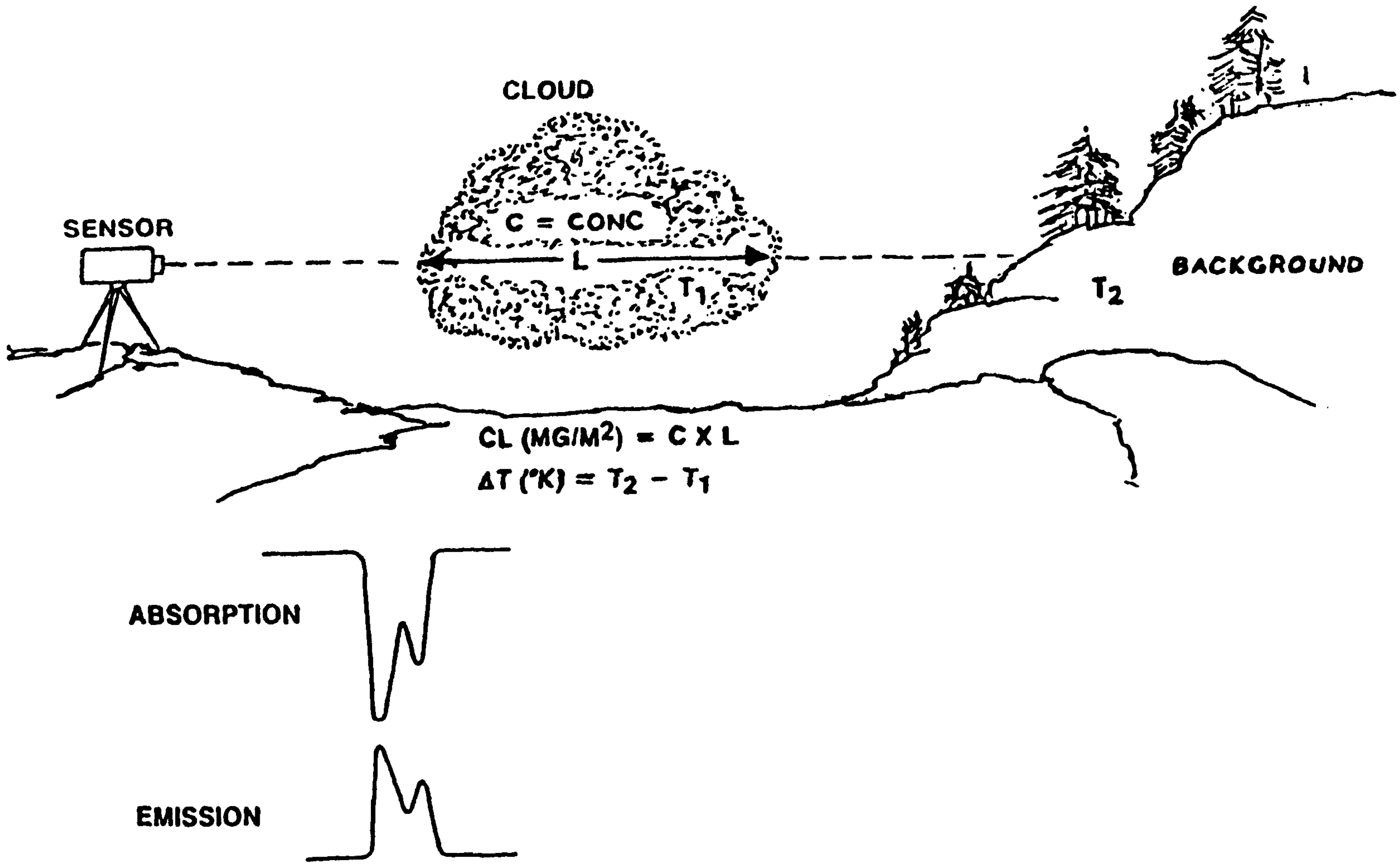

Figure 1. Operation of a Passive Infrared Remote Sensor

Real-time implementations of digital filter and pattern recognition interferograms are a necessity for passive-remote sensing measurements. Open-path measurements can be enhanced by the ability to notify an operator of the presence of a particular chemical vapor. A feedback mechanism provided by real-time software assists the operator in identifying spectral contributions from a vapor cloud before the conclusion of a particular experiment. The real-time data collection and analysis software enables the operator to make adjustments or modifications to the experimental design to improve data collection results from a field trial.

This study describes the computer software implementation of a real-time detection algorithm on a personal computer (PC) that has been integrated into a commercial interferometer hardware package. Data collection programs are described that enables the user to collect interferograms for later analysis. The data collection and analysis program names are listed in Table 1.

Table 1. Overview of Program Names

Name	Program Discussion
midcol	collects data to disk from the midac unit
replay	replays and displays collected risk data
mtrx	collects data from the midac and runs the matrix filter and pattern recognition program
mtrxd	reads data from disk and runs the matrix filter and pattern recognition program
matgraph	display the result file processed by programs mtrx and mtrxd
convintf	converts interferogram files created by the program midcol to a format that can be read by SPECTRACALC

2. EXPERIMENTAL PROCEDURES

Data collection and real-time data analysis software was developed for a commercial interferometer manufactured by Midac Corporation (Irvine, CA). This small, low-cost FTIR remote sensor can be configured as a passive, single-ended, or bi-static active system. The system can be used in an active mode that includes a parabolic mirror and an IR source. Active source configurations require the addition of a telescope allowing the field-of-view of the interferometer to be fully filled by the source for long path length applications. The passive arrangement (Figure 2) used for data collected for this study can be used either with or without a telescope. A telescope is not needed if the target chemical vapor cloud is sufficient to fill the field-of-view of the interferometer.

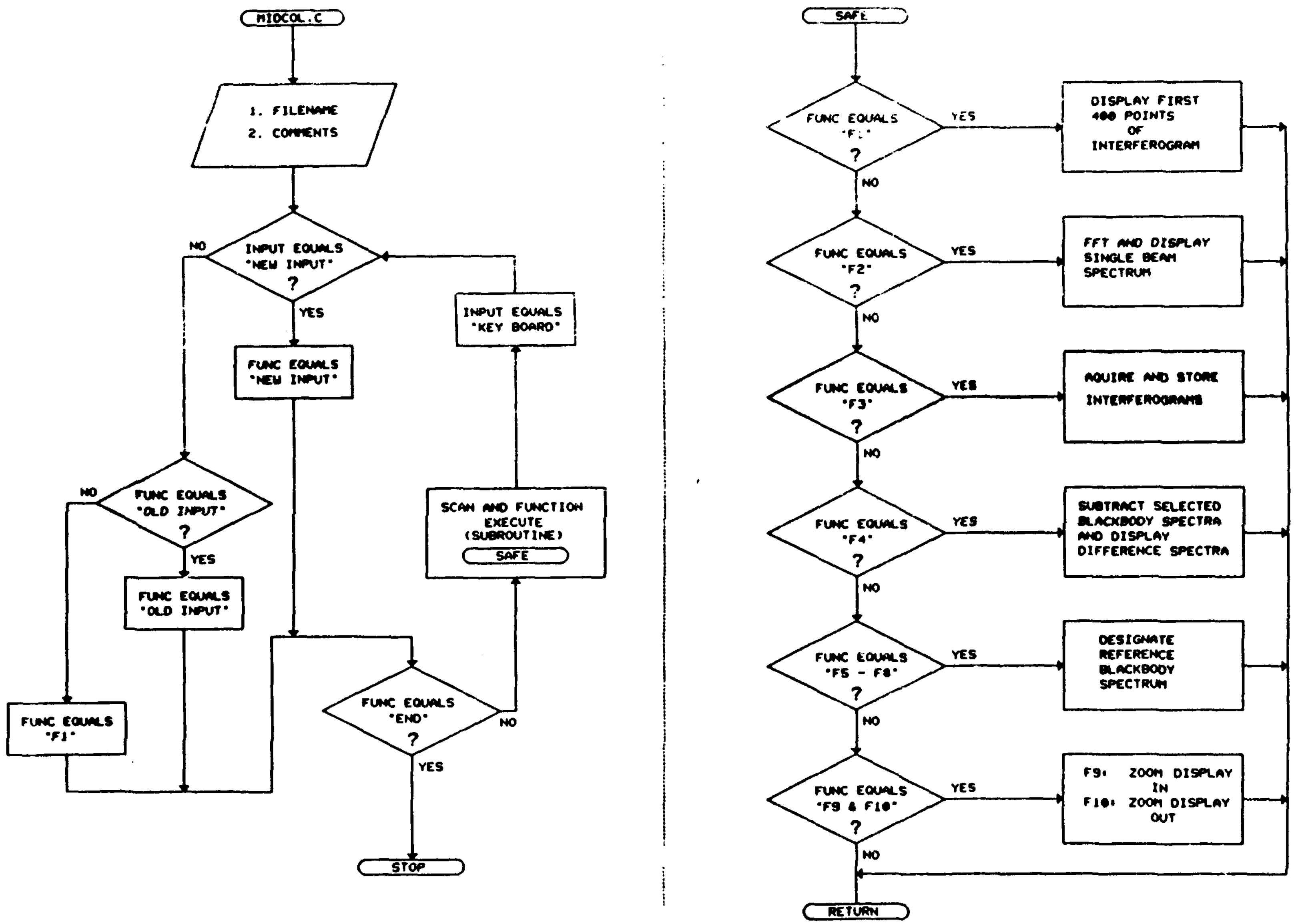

Figure 2. Data Collection Program Flow Chart

The Midac spectrometer consisted of a Michelson linear drive interferometer that was specifically optimized for the 8 to 12 μm wavelength atmospheric window. The interferometer had an optically coated nonhygroscopic front window with an optical diameter of 1.75 in. The interferometer beamsplitter was constructed of zinc selenide (ZeSe) and had a tolerance of better than 1% deviation from the 50% transmission optical requirement between the wavelength region of 7 and 14 μm. The IR energy exiting the interferometer was converged onto a 1 by 1 mm MCT detector element by a gold coated off-axis parabolic mirror. The detector element was mounted in a dewar and cooled with liquid nitrogen.

Analog data was amplified by a low-noise pre-amplifier and a low-noise, high-gain post amplifier. The analog signal was passed to a 16-bit analog-to-digital converter for digitization. The digitization rate of the analog-to-digital converter was controlled by a reference He-Ne laser and detector arrangement that included two visible detectors that also monitors the direction of mirror travel.

The interferometer hardware had the capability of scanning interferograms as short as 512 points or as large as 16 K points. In addition, the servo scanning hardware had the capability of sampling every He-Ne laser zero crossing or sampling as few as only every eighth laser zero crossing. The hardware servo sampling flexibility allows one to select an optimal sampling rate and resolution for a given open-path experiment. The software that was developed had the capability of collecting data at all of the sampling rates and resolutions that the interferometer hardware allowed.

3. DATA COLLECTION PROGRAM DESCRIPTION

Data collection software was completed in "C" language to allow for the storage and display of interferograms obtained from the Midac spectrometer. Version 3 of the software allows collection of interferograms at various resolutions and sampling rates for a display of the Fourier transformed spectrum. Two data collection programs allow for the data collection to disk and also the capability to display interferograms at a later time.

The interferometer data collection program (midcolv) enables the user to store up to 3000 sequentially collected interferograms into a single disk file. This collection program has the ability to Fourier transform the interferogram, background subtract a reference spectrum, or compute a calibrated blackbody reference spectrum in real-time to yield a corrected difference emission profile. Conversion from any one data display type to another is achieved by using single key strokes. A single keyboard control function is useful for outdoor remote sensing field applications where the use of a mouse or window display menus becomes difficult to manipulate. A flow chart of the data collection program is shown in Figure 2.

The data collection program has eight screen display options that are enabled with the use of function keys F1 through F8. Function key F1 is defaulted to display the first 400 points of a collected interferogram. The

interferogram screen display can be expanded or contracted to allow for more or less points to be displayed. The expansion and contraction functions are enabled by the use of four functions keys which are F9, F10, the left arrow key, and the right arrow key. Function key F9 enables one to contract the interferogram screen, while function key F10 enables an expansion of the screen display. The left and right arrow keys enable one to rotate data on the display screen to the left or right. By using all four of the single function keys, one can display any range of interferogram points without the need of a special keypad, joystick, or mouse control. These keyboard control functions greatly simplify operation of an FTIR sensor in an outdoor field monitoring application.

Function key F2 enables one to view the Fourier transformation of each collected interferogram. The program default is to display the entire range of the spectrum. This default display range can be expanded using function key F9 to display a selected wavelength range. The function keys, F10, left arrow, and right arrow, can be also be used to contract or rotate the display of the transformed single-beam spectrum.

Function key F3 is used to collect interferograms and store data to a disk file. This display option lists the name of the file, the last interferogram number stored to disk, and whether a known data collection error has occurred. A list of error codes is provided in Table 2. Provisions exist in the software that allow the user to include additional codes in the error table. The program automatically uses the error table to scan a particular interferogram for storage in the scan header and display. The addition of the error code table is extremely useful for laboratory and outdoor passive-remote sensing measurements, because an operator is notified instantly of a possible error condition during data collection. By displaying the error code, the operator can correct a possible problem before the end of the open-path measurement.

Table 2. Error Codes

Error Code Number	Error Code Meaning
1	less than correct # of data points collected by computer for scan
2	A/D overflow
3	centerburst interferogram point position change from last scan
4	centerburst position out of range
5	low signal (A/D gain very low)

Interferogram data stored to disk using function key F3 is formatted as shown in Table 3 and Table 4. This standardized data format contains a global header, an interferogram subfile header, and the

Table 3. Data Collection Disk Format

Byte Position	Bytes Used	Field Description	Data Type
		GLOBAL HEADER - 512 bytes	
General info			
0	10	filename (CCC####)	string
10	10	date (MM/DD/YY)	string
20	10	start time (HH:MM:SS)	string
30	10	stop time (HH:MM:SS)	string
40	2	stop scan number	int
42	10	operator's name	string
52	44	unused	string
Sensor information			
96	20	sensor identification	string
116	2	collection mode	int
118	2	integer type	int
120	2	points per scan	int
122	8	resolution	double
130	8	scan speed	double
138	8	mirror velocity	double
146	8	sampling frequency	double
154	8	starting frequency	double
162	8	ending frequency	double
170	8	maximum wavenumber sampled	double
178	2	# of zero crossings per sampled point	int
180	16	unused	string
Weather information			
196	2	ambient temperature	int
198	8	barometric pressure	double
206	2	humidity	int
208	2	wind speed	int
210	2	wind direction	int
212	2	sensor direction	int
214	2	precipitation code	int
216	40	unused	string

Table 3. Data Collection Disk Format (Continued)

Byte Position	Bytes Used	Field Description	Data Type
	Comments		
256	64	comment - line #1	string
320	64	comment - line #2	string
384	64	comment - line #3	string
448	64	comment - line #4	string

SCAN HEADER - 64 bytes

Byte Position	Bytes Used	Field Description	Data Type
0	2	scan number	int
2	10	filename	string
12	10	time (HH:MM:SS)	string
22	2	centerburst location	int
24	2	Analog to Digital gain setting	int
26	2	number of scans co-added	int
28	34	unused	string
62	2	error	int

SCAN DATA - variable bytes

Data block will consist of integer data of type designated in the global header, stored contiguously for length specified by the points per scan as also designated in the global header.

Table 4. SpectraCalc Data Format

DATA BYTE OFFSET	TYPE	EXPLANATION
0	byte	version number (optional)
1	byte	must be 4D hex
2	word	exponent for all y values
4	float	number of y data points
8	float	x value of the first data point
12	float	x value of the last data point
16	byte	type of x values: 0 = arbitrary 1 = CM-1 (wavenumbers) 2 = micrometers (μm) 3 = nanometers (nm) 4 = seconds (sec) 5 = minutes (min) 6 = hertz (Hz) 7 = kilohertz (kHz) 8 = megahertz (MHz) 9 = mass units 10 = parts per million (ppm) 255 = double igram (not labeled)
17	byte	type of y values: 0 = arbitrary 1 = interferogram 2 = absorbance 3 = kubelka-munk 4 = counts or CPM 5 = volts or KEV 6 = degrees 128 = transmission 129 = reflectance (values 0-127 must have positive peaks) (values 128-255 must have negative peaks) year that data was collected
18	word	if 0 then date/time is ignored month that data was collected
20	byte	day of month data was collected
21	byte	hour that data was collected
22	byte	minute of hour data was collected
23	byte	resolution of data in ASC text
24	8 bytes	reserved for 8 floating values
32	32 bytes	(the first word is the peak position) ASC II comments (comment must end with a zero
64	192 bytes	byte) y signed 32 bit word reversed two's complement
256		reversed fractions

interferogram data. The interferogram data format allows for storage of data points in 8, 16, 32, or 64 bit data representations. Data type information is stored in the global header. The current implementation of the MIDAC software uses a 16-bit data storage to correspond with the number of bits of the analog-to-digital converter.

The global header is divided into general information, sensor information, weather information, and comment information. The addition of weather information for outdoor passive-remote sensing measurement is almost essential for some data collection exercises. Information about the other sensor parameters includes the collection mode, the data type, the scan speed, the resolution, the sampling rate, and the number of zero crossings to sample for each data point.

The scan header contains information that is specific to each collected interferogram. Included in the scan header is the interferogram scan number, the time the interferogram scan was collected, the centerburst point position, the analog-to- digital converter gain setting, the file name, and the number of interferogram scans that were coadded upon data storage. The time clock on the PC is queried before the start of each interferometer scan. This information is converted to ASC format and written as part of the header information for each interferogram scan.

An additional scan header descriptor, which is included in the data format, is the file name information. This data field enables the user to tag the source of single interferograms in the large single file storage format. This descriptor is extensively used in the conversion of data types between the commercial SpectraCalc software (Galactic Industries, Salem, NH) and this remote sensing data format.

Function key F4 in the data collection program is used to display a difference spectrum in real-time on the screen. In this data collection mode, a background interferogram is collected, transformed, and subtracted from all following interferogram scans. The result is a display of the difference spectrum that shows the change in thermal emission profile as a function of time. This display function is extremely useful for open-path optical alignment measurements in which the instrument, vapor cell, and the IR source can be simultaneously adjusted.

Function keys F5 through F8 are used to subtract previously stored calibrated reference spectra from each of the collected transformed input interferograms. When one of these function keys is pressed, the program reads a corresponding disk file name, collects an input interferogram, and displays the result of a spectral subtraction. Up to four different calibrated single-beam reference spectra can be stored and recalled during data collection. It is recommended that the calibration reference spectra be transformed and stored to disk through the use of the commercial Spectracalc program.

4. DATA CONVERSION PROGRAMS

Utility data format conversion programs are an essential requirement for any passive-remote sensing application software package, because several excellent commercial software packages exist that aid in the data analysis. Two computer programs were developed that allow conversion of the data format shown in Table 3 and Table 4 to the standard commercial Spectracalc data format. Remote sensing interferogram data collected with the data collection software can be converted, processed, displayed, and output to a hardcopy device through the use of this commercial software.

The first program converts a specified number of remote-sensing interferograms into a single SpectraCalc format interferogram file. The program requests the user to identify a particular group of interferograms on disk to convert. The program reads each remote sensing interferogram and converts the data to a two's complement binary representation. Data stored on the PC disk has a file name that provides a reference index name that is keyed to the source interferogram number.

A second data conversion program converts multiple SpectraCalc files into the single-remote sensing interferogram file format (Table 3). The SpectraCalc data format does not normally include some of the remote sensing header parameters (i.e., weather information and sensor positioning parameters) for data storage of laboratory measurements. Therefore, this program requires the user to input a number of global header parameters describing the data type for storage of a disk file.

5. DATA ANALYSIS PROGRAMS

Real-time data analysis computer programs have been developed for the Midac interferometer that apply a set of automated pattern recognition methods for the detection of specific chemical vapor species. Fourier transform infrared passive remote sensing applications require that a reference background spectrum be subtracted from open-path field spectra to remove the instrument and background spectral contributions. For many reasons, it is not always possible to obtain a true reference spectrum during passive remote sensing data collections. One practical reason of not being able to obtain a true spectrum is that field sites to be monitored may already be contaminated with the vapor species to be measured. In this case, because the vapor species is always present, one may not be able to obtain an accurate emission reference spectrum that does not also contain the vapor species to be monitored.

During the last several years, research has been conducted to develop a signal processing method for eliminating the need of a background reference spectrum. Digital filter and pattern recognition methodology for automatic detection of atmospheric species was developed that eliminates this background reference problem.[1-5] The methodology uses a preprocessing strategy called a "matrix filter" and a multiple linear discriminant pattern recognition method to discriminate against various backgrounds. Collected interferogram data is used as the input for the generation of an optimal set

of digital filter and multiple discriminate pattern recognition coefficients. Optimized coefficients are generated based on the number of interferograms classified correctly from a training data set. The reader is referred to the open literature publications referenced that describe the mathematical background for the generation of the digital filter and pattern recognition coefficients. (1-4)

Implementation of the mathematical method described in the references is shown in Figure 3. The computer program enables interferogram data to be collected directly from the interferometer and processed through the algorithm. Implementation of the algorithm requires the storage of one set of digital filter coefficients for each point along the digitized interferogram. A current implementation for the acetone vapors detection requires an average of 17 coefficients from each interferogram point along a 75 point interferogram segment length. Data storage requirements for the acetone filter and pattern recognition coefficients were less than 7 K 32 bit words of storage. One set of digital filter and pattern recognition coefficients is required for identification of each particular vapor species.

Current execution of the computer program on a Dell 486 50 MHz PC allows for the real-time identification of acetone without the need of a reference background subtraction. Each interferogram is segmented, filtered, multiplied by the multiple linear discriminants, and subsequently summed to give a single discriminator output result. The discrimination and data display to the screen is completed in less than 20 msec for a single interferogram scan. As an example, if one uses this implementation of software, then enough processor capability is available, using a moderately fast 486 PC, to simultaneously collect and monitor for the presence of up to five compounds. The software on the 486 PC can identify the five different vapor species on every interferogram at scan rates of up to 5/scans/sec. For Acetone and 2-Butanone, the correct classification percentages have been reported to be over 99%.[6]

Figures 3 and 4 show the real-time computer screen display of two data collection runs. The plot in Figure 3 is a data run that shows the response of the discriminator as a function of time. The x-axis shows the interferogram number collected, while the y-axis is the relative output response of the discriminator. The plot indicates that acetone was present between the collection of interferograms 98 to 143 and also 155 to 189. All of the other interferograms outside of these ranges show that acetone was not present in the field-of-view of the interferometer. The plot in Figure 4 is a collection of 1550 interferograms that were collected when no acetone was present in the field-of-view of the interferometer. All of the individual interferogram discriminant responses are below the zero threshold indicating that no acetone is present.

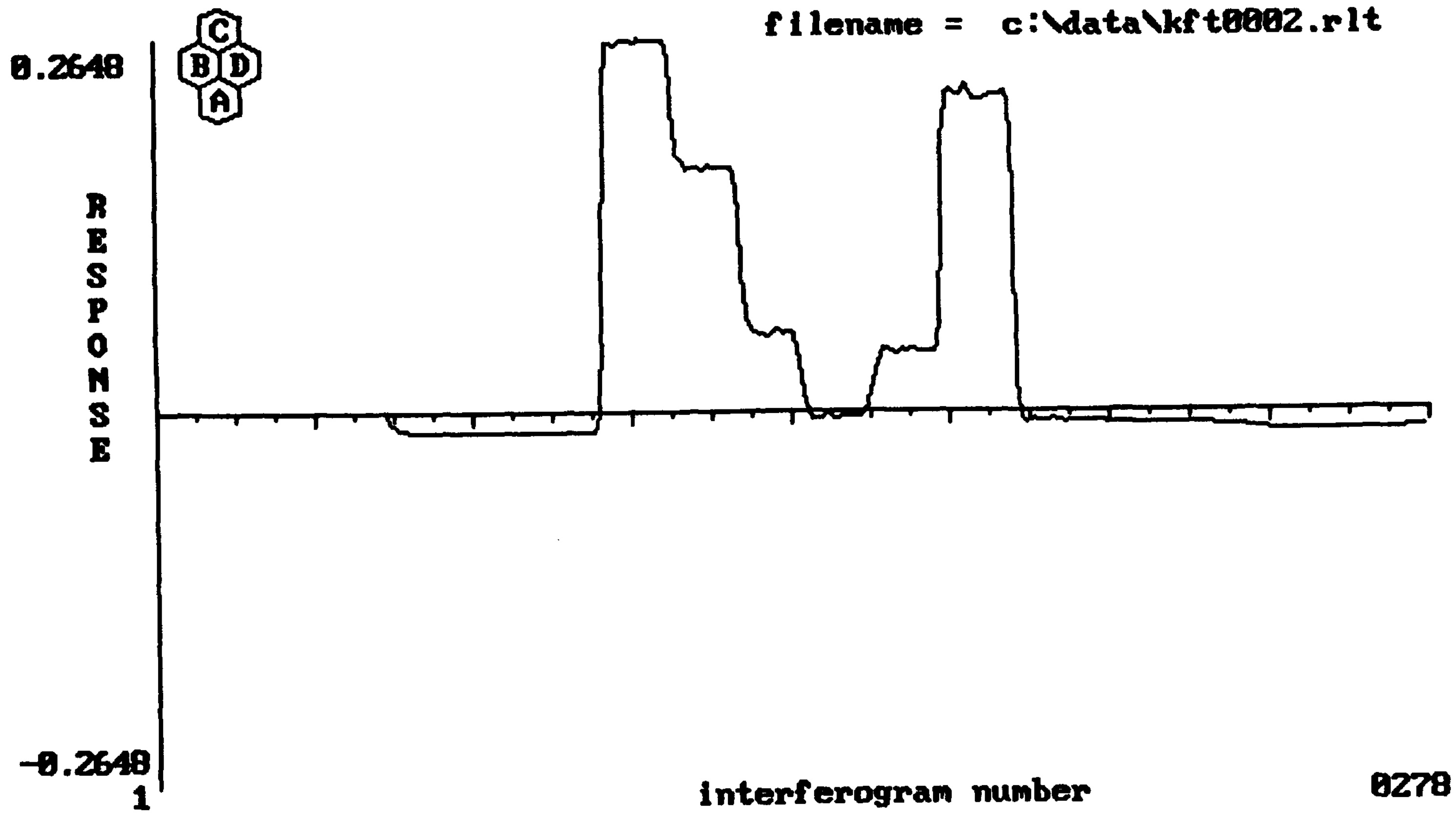

Figure 3. Pattern Recognition Results from the Real-Time Computer Program

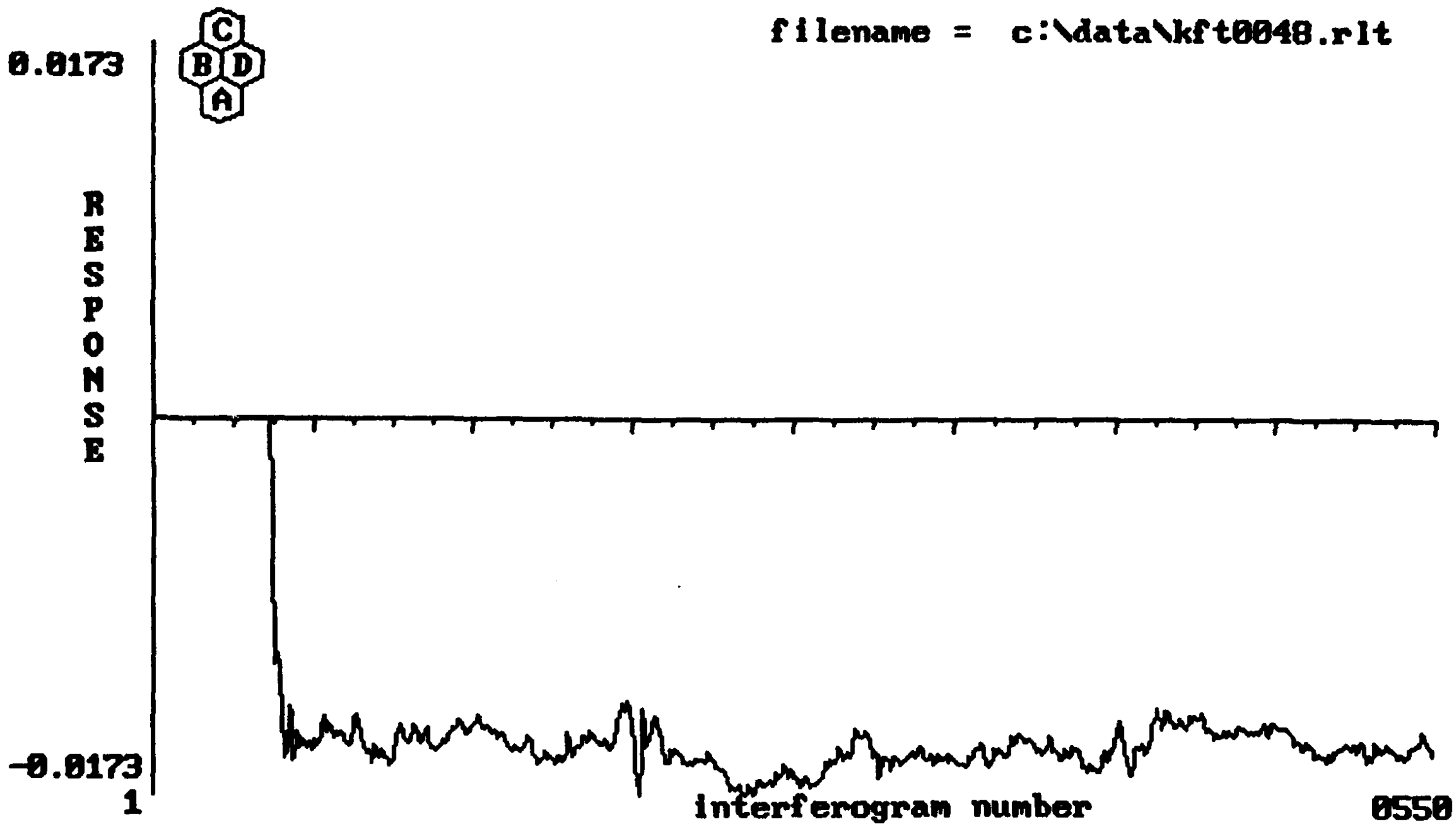

Figure 4. Pattern Recognition Results for a Background Data File

6. CONCLUSIONS

Passive Fourier transform infrared remote sensing data collection and real-time analysis software has been completed for a commercial interferometer that eliminates the need of a background reference spectrum for open path measurements. Data collection routines written in "C" language code allow the user to collect and display interferograms for later analysis. Data analysis routines were completed to allow for the real-time detection of a particular chemical species from single-scan interferograms using a low-powered IBM PC interfaced to a commercial interferometer. Pattern recognition coefficients for the direct interferogram identification and detection of chemical vapors have been developed and executed on a single scan in less than 20 msec using an IBM-PC 486 50 MHz computer.

Blank

LITERATURE CITED

1. Small, G.W.; Kroutil, R.T.; Ditillo, J.T.; and Loerop, W.R., "Detection of Atmospheric Pollutants by Direct Analysis of Passive Fourier Transform Infrared Interferograms," Anal. Chem. Vol. 60, pp 264-269 (1988).

2. Small, G.W.; Harms, A.C.; Kroutil, R.T.; Ditillo, J.T.; and Loerop, W.R., "Design of Optimized Finite Impulse Response Digital Filters for Use with Passive Fourier Transform Infrared Interferograms," Anal. Chem. Vol. 62, pp 1768-1777 (1990).

3. Kroutil, R.T.; Ditillo, J.T.; and Small, G.W., "Signal Processing Techniques for Remote Infrared Chemical Sensing," Computer-Enhanced Analytical Spectroscopy, Chapter 4, Vol. 2, pp 71-111, H.C.L. Meuzelaar, Ed., Plenum Press, New York, NY, 1990.

4. Kaltenbach, T.F.; and Small, G.W., "Development and Optimization of Piecewise Linear Discriminants for the Automated Detection of Chemical Species," Anal. Chem. Vol. 63, pp 936-944 (1991).

5. Small, G.W.; and Barber, A.S., Application of Digital Filtering and Pattern Recognition Techniques to Interferogram-Based Fourier Transform Infrared Qualitative Analysis: Investigation of Spectral Interferences," Chemom. Intell. Lab. Syst. Vol. 15, pp 203-217 (1992).

6. Kroutil, R.T.; Combs, R.J.; Knapp, R.B.; and Small, G.W., "Remote Infrared Vapor Detection of Volatile Organic Compounds," In SPIE Proceedings of the 9th International Conference on Fourier Transform Spectroscopy, Calgary, Canada, 1993, UNCLASSIFIED Report.

Blank

APPENDIX A

DESCRIPTION OF COMPUTER PROGRAM OPTION LIST

MIDAC DATA COLLECTION PROGRAM OPTIONS

```
-----------------------------------------------------------------
program MIDCOL  ---  Midac data collection program operation

(A) To start program type:  midcol/f   (to collect up to 550
                                         interferograms to a single
                                         file)
                                or

                            midcol     (to collect up to 3000
                                         interferograms to a single
                                         file)

(B) Program input:   The program will request the following:

                  (1) a filename to store the data
                  (2) 4 lines of input to store comments

(C) The following keys are used to control the program:

   KEY                                Program Action
-----------                   ---------------------------------

   F1 (function key #1)        display the collected interferogram
                               to the screen
   F2 (function key #2)        display the collected spectrum
                               to the screen
   F3 (function key #3)        collect interferograms to a disk
                               file
   F4 (function key #4)        collect a background spectrum
                               and subtract all following
                               spectra
   F5 (function key #5)        subtract transformed interferogram
                               from spectral disk file f5.fsp
   F6 (function key #6)        subtract transformed interferogram
                               from spectral disk file f6.fsp
   F7 (function key #7)        subtract transformed interferogram
                               from spectral disk file f7.fsp
   F8 (function key #8)        subtract transformed interferogram
                               from spectral disk file f8.fsp
   F9 (function key #9)        contract the interferogram display
   F10 (function key #10)      expand the interferogram display
   left arrow key              move the interferogram to the left
                               on the display screen
   right arrow key             move the interferogram to the right
                               on the display screen
   End (the key labeled "End")  quit program

-----------------------------------------------------------------
```

MIDAC DATA COLLECTION DATA DISPLAY PROGRAM

---

program REPLAY -- This program will replay the data collected by the program MIDCOL.

(A) To begin program type: replay filename (the program name and a data filename is required)

example: replay c:\temp1.dat

(B) The following keys are used to control the program:

KEY	Program Action
F1 (function key #1)	display the collected interferogram to the screen
F2 (function key #2)	display the collected spectrum to the screen
F4 (function key #4)	display a background spectrum and subtract all following spectra
F5 (function key #5)	subtract transformed interferogram disk file f5.fsp
F6 (function key #6)	subtract transformed interferogram disk file f6.fsp
F7 (function key #7)	subtract transformed interferogram disk file f7.fsp
F8 (function key #8)	subtract transformed interferogram disk file f8.fsp
F9 (function key #9)	contract the screen display
F10 (function key #10)	expand the screen display
left arrow key	move the screen display to the left
right arrow key	mode the screen display to the right
PGUP key	speed up the screen display
PGDOWN key	slow down the screen display
End (the key labeled "End")	quit program

PROGRAM CONVINTF - UTILITY DATA CONVERSION

---

program CONVINTF --- utility data conversion program

(a) To start the program type: (1) go to the data directory in the file to convert is located
(2) TYPE: convintf

example: if the data is a file called nct0018.dat and located in a directory called "c:\data" and the source code is in a directory called "c:\midac\collect", then one would go to the "c:\data" directory and type---

```
c:\midac\collect\convintf
```

(b) Program input: The program will request the following information:

(1) a midcol collected interferogram file to read
(2) a partial character name to append on the file name for the output files.
(3) the starting interferogram number to read
(4) the ending interferogram number to read
(5) the format type - 0 = floating point format
1 = SpectraCalc format

example:

(1) Input the interferogram file name to read: nct0018.dat
(2) Input a partial character name for output interferogram filename : aaa
(3) Input the starting interferogram number to convert: 20
(4) Input the ending interferogram number to convert: 30
(5) Input the data format type: 0

Result:
Ten data files will be created in the directory "c:\data" that will be called aaa0020.fsp to aaa0030.fsp. These file will contain single interferograms that can be read by the program SPECTRACALC using the import command.

Blank

APPENDIX B

MIDAC SERVO CONTROL JUMPERS

In order for the Midac data collection program to operate in a correct manner, the servo-board on the Midac interferometer must be jumpered in the following manner.

If one is looking at the back of the Midac board titled "Combo Digital Mirror Drive" board, then one can find a row of jumper pins labeled J5 (on the left) to J1 (on the far right side of the board). The basic purpose of each set of jumper is as follows:

- J5 block - determines if the system scans using either the high speed or low speed electronics
- J4 block - controls the clock oscillator on the board to determine the sampling frequency (by selecting options for both J5 and J4 one can select a wide range of scan speeds)(the range is from 20 KHz to 160 Khz sampling frequency)
- J3 block - these jumpers control the number of fringes for every sampled point. The jumpers allow one to sample a point for every He-Ne laser fringe up to the case of 8 zero crossings for every sampled point.
- J2 block - this block of jumpers sets the scan length of the interferometer. The selectable range of scan lengths correspond from 1/2 to 32 wavenumbers of resolution. The software package SPECTRACALC can use any of the scan lengths up to the maximum value jumpered by J3. For example, if 4 wavenumbers were selected, then SPECTRACALC would be able to collect all resolutions up to 4 wavenumbers (32,8,4). It would not be able to collect resolutions at 2,1, and 1/2 wavenumber.
- J1 block - this set of jumpers delays the start of scan time. These jumpers are used to place the centerburst at various places in the sampled interferogram segment. This set of jumpers allows one to collect either single-sided or double-sided interferograms.

The exact jumper settings are as follows:

J5 - high/low speed option

pin location	label	meaning
left pin	L	selects the low speed scan option
middle pin	H	selects the high speed scan option
right pin	SW	if this is jumpered, then the switch labeled "SW2" on the "Switchable ADC board" can be used to select the scan speed

J4 - controls sampling speed

pin location	label	meaning
left pin	pins 1&2	20 Khz sampling frequency
second pin	3&4	40 KHz
third pin	5&6	80 KHz
right pin	7&8	160 Khz

J3 - number of zero crossings per sampled point

pin location	label	meaning
left pin	M1	samples every 4th zero crossing
second pin	M2	samples every 8th zero crossing
third pin	L1	samples every other zero crossing
right pin	2L	samples every zero crossing

J2 - selects the scan length from 1/2 to 32 wavenumbers

This block of jumpers selects the scan length. The scan length jumpers are offset by one pin to the left set as compared to the bottom set. The right most possible jumpers select the lowest resolution while the far left jumpers set the highest resolution.

top set of pins	# of points collected (for L1 jumper)	scan resolution
right pins	no option	
second pins	32 K	1/2 wavenumber
third pins	16 K	1 wavenumber
fourth pins	8 K	2 wavenumbers
fifth pins	4 K	4 wavenumbers
sixth pins	2 K	8 wavenumbers
seventh pins	1 K	16 wavenumbers
eight pin	1/2 K	32 wavenumbers
left pins	no option	

J1 - selects the scan delay from the zero path difference

DLY4 is the most significant delay bit while the DLY1 jumper controls the least significant delay bit. The jumper positions for the up position is a "0" or "small delay" while the down position is a "1" or a "large delay".

The switch on the "Switchable ADC board" located at the back of the midac unit are as follows:

switch labeled SW1 --- gives one the ability to connect an external detector.
position up - use internal MCT detector
position down - use an external detector
switch labeled SW2 --- controls the scan speed (if the SW jumper is selected on block J5) This switch

will alternate between different sets of analog filters on the ADC board.

The recommended set of jumper selections of the servo board for use with the data collection program "midcol" is as follows:

J5 - H or L   high speed or low speed scanning options
J4 - 1,2   20 KHz sampling (recommended)
J3 - M2   sampling every 8th zero crossing
J2 - on top - 5th pin from left   4 wavenumber resolution
on bottom - 6th pin from left
J1 - delay so that the centerburst is between 40 - 90 points from the start of scan
SW1 - use the internal MCT detector - position "up"
SW2 - switch position "up"

Midac PC interface board jumper setting:

jumper J1 - set jumper to IRQ2

Blank

APPENDIX C

REPLAY PROGRAM

```
/************************ program REPLAY ********************************/
/*
 program REPLAY                          version 4.0

  This program is used to read interferogram data from disk,
  display, and FFT for display. This program will be used
  for data collection for the Midac interferometer.

  author: Bob Kroutil

  date: June 1992

  routines called:
     plotr     - plots an interferogram or spectrum
     logoega   - prints the CRDEC logo
     draw_axis - draws the axis for the plots for either interferogram
                 or spectra
     cmpfft    - computes the fast Fourier transform
     norml     - normalizes the spectrum
     Microsoft C graphics routines

-----------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <graph.h>
#include <math.h>
#include <dos.h>
#include "exreplay.def" /* external definitions for the program replay */
#include "headers.def"/* contains the interferogram data structure  */

/* The following global parameters are the following:
     MAXPOINTS= the number of interferogram points
     GH_LIMIT = the number of bytes in the global interferogram header
     SH_LIMIT = the number of points in the subfile interferogram header
     FEND      = the key code to exit the program
     FRIGHT    = the key code to expand the interferogram display
     FHOME     = the key code to reset the interferogram display
     FLEFT     = the key code to compress the interferogram display
     FSEL5     = the key code to background subtract disk file f5.fsp
     FSEL6     = the key code to background subtract disk file f6.fsp
     FSEL7     = the key code to background subtract disk file f7.fsp
     FSEL8     = the key code to background subtract disk file f8.fsp
     FINT      = the key code to display interferograms
     FSPEC     = the key code to display spectra
     FDIFF     = the key code for the spectral background subtraction
     ROLLL     = the key code to roll the display data to the left
     ROLLR     = the key code to roll the display data to the right
     MDLY      = increase the screen display time
     LDLY      = decrease the screen display time
*/
```

```
#define GH_LENGTH 512
#define SH_LENGTH 64
#define FEND 79
#define FRIGHT 68
#define FLEFT 67
#define FSEL5 63
#define FSEL6 64
#define FSEL7 65
#define FSEL8 66
#define FINT 59
#define FSPEC 60
#define FDIFF 62
#define ROLLL 75
#define ROLLR 77
#define MDLY 81
#define LDLY 73

main(argc,argv)
int argc;
char *argv[];

{
/* The following parameters are the following:
    raw_buf          - the interferogram buffer (real values)
    spc_buf          - the complex interferogram buffer, also used as a
                       work array
    pi               - value of the constant pi
    raw_data         - the interferogram buffer (integer values)
    scan             - the scan number
    index            - an indexing variable
    fp1              - file open variable
    istps            - beginning point in array to display to screen
    iendp            - ending point in array to display to screen
    ichng            - the number of points to roll or expand screen
    spoints          - the maximum number of points to display
    imode            - 0=display interferogram, 1=display spectrum
    loop             - graphics display page
    ch               - used for an input
    spc_bak          - the background spectrum array
    bkgr             - the background flag
    sstart           - the starting point for difference spectrum
                       display
    send             - the ending point for difference spectrum
                       display
    dpoints          - the number of points to display on the difference
                       spectrum
    ercod            - interferogram error code
    buffer           - working character array for output
    lastpeak         - previous interferogram center burst array
                       position
    burst            - array position in subfile header for center burst
    global_header,gh - the global header structure
```

```
    scan_header,sh    - the subfile header structure
*/

  void plotr(), logoega(), draw_axis(), cmpfft(), normal(), getspc();
  int errcod();
  float pi, minx_val, maxx_val, miny_val, maxy_val;
  int bkgr, ercod, lastpeak, burst, istps, iendp, ichng, limit, slimit;
  int scan, index, fp1, spoints, imode, loop=0, jndex=1, idly;
  long int pktopk, max_val, min_val;
  char ch, buffer[80];
  extern float raw_buf[], spc_buf[], spc_bak[];
  extern int raw_data[];
  struct global_header gh;
  struct scan_header sh;
  union REGS inregs;   /* REG structure for timing input */
  union REGS outregs;  /* REG structure for timing output */

/* check to see if a data filename was given */
  if (argc != 2)
    {
    printf("\nUsage: replay infile\n");
    exit(1);
    }

  /* Open a file connection to the Midac data file */
  if ((fp1 = open ( argv[1], O_RDONLY|O_BINARY)) < 0)
    {
     printf ("\n\"MIDCOL\" is unable to open %s\n",argv[1]);
     exit(1);
     }

  /* Set up the screen */
   _setvideomode (_ERESCOLOR);
   _setbkcolor (_BLUE);

  /* read in the global header */
  read (fp1, &gh, GH_LENGTH);

  /* find the number of points in the interferogram file - exit if too large
*/
  limit = gh.scan_size;
  slimit = limit / 2;
  if (limit > MAXPOINTS)
    {
     printf("\nERROR: > %d number of points in file. #points= %d\n",
            MAXPOINTS, limit);
    }

  /* set the other parameter values */
  pi=4.*atan(1.); /* the value of pi */
  istps = 1;        /* the starting point number to display on the screen */
  iendp = 400;      /* the ending point number to display on the screen */
```

```
  ichng = 50;        /* the number of points to expand or roll screen */
  spoints = limit; /* the maximum number of points that can be displayed */
  imode=0;          /* 0=,display interferogram ; 1=display spectrum */
  bkgr = 1;          /* set the background flag to collect */
  idly = 5;          /* set the delay timer for interferogram display */

  /* loop to get each interferogram to process  */
  for (scan=0; scan <= gh.stop_scan; scan++)
   {

  /* add in a delay if desired --- comment this out if running
                                    on a 386 system */
     inregs.h.ah = 0x86;   /* delay service */
     inregs.x.cx = idly;      /* set the high order delay word */
     inregs.x.dx = 0;      /* set the low order delay word */
     int86(0x15,&inregs,&outregs); /* call the ROM BIOS timer delay service */

/*------------------- select user mode -----------------------------*/
     if (kbhit() != 0)  /* check to see if a key was pressed */
       {
        ch=getch();
        if (ch == FEND)  /* exit program */
          {
           close (fp1);
           _setvideomode(_DEFAULTMODE);
           exit(1);
           }
        if (ch == FRIGHT) /* expand the screen display */
          {
            iendp = iendp - ichng;
            istps = istps + ichng;
            if (istps >= iendp)
              {
                istps = istps - ichng;
                iendp = iendp + ichng;
              }
          }
        if (ch == FLEFT)/* contract the screen display */
          {
           iendp = iendp + ichng;
           istps = istps - ichng;
           if (istps < 1 ) istps = 1;
           if (iendp > spoints) iendp = spoints;
          }
        if (ch == ROLLR)  /* roll the data to the right */
          {
           iendp = iendp - ichng;
           istps = istps - ichng;
           if (istps < 1)
             {
              istps = 1;
              iendp = iendp + ichng;
```

```
           }
         }
        if (ch == ROLLL)  /* roll the data to the left */
          {
           iendp = iendp + ichng;
           istps = istps + ichng;
           if (iendp > spoints)
             {
              iendp = spoints;
              istps = spoints - ichng;
             }
          }
       if (ch == FINT)/* display interferogram */
         {
          imode=0;
          istps = 1;
          iendp = 400;
          spoints = limit;
         }
       if (ch == FSPEC)/*  display spectrum */
         {
          imode=1;
          istps = 1;
          iendp = slimit;
          spoints = iendp;
         }
       if (ch == FDIFF)  /* display difference spectrum */
         {
          imode = 3;
          bkgr = 1;
          istps = 181;
          iendp = 363;
          spoints = slimit;
         }
      if (ch == FSEL5 || ch == FSEL6 || ch == FSEL7 || ch == FSEL8)
         {
          imode = 3;
          bkgr = 0;
          istps = 181;
          iendp = 363;
          spoints = slimit;
          getspc (spc_bak, spoints, ch);
         }
      if (ch == MDLY)
          idly = idly + 5;
      if (ch == LDLY)
         {
          idly = idly - 5;
          if (idly < 0)
            idly = 0;
         }
    }
```

```
/*------------------------------------------------------------------*/

  /* read in the subfile header interferogram information from disk */
  read(fp1, &sh, SH_LENGTH);

  /* read in the interferogram data from disk */
  read (fp1, raw_data, limit*2);

  /* find the number of points in the file to display */
  limit = gh.scan_size;

  /* convert the integer array to an ungain ranged floating array */
  for (index = 0; index < limit; index++)
   raw_buf[index+1] = (float) raw_data[index];
  raw_buf[0]=0.0;
  spc_buf[0]=0.0;

/* find the center burst in the data header */
  burst = sh.peak_location - 1;

/* do the Fourier transformation if IMODE=1, set by F1 key */
  if (imode == 1)
   {
     cmpfft(raw_buf, spc_buf, limit, pi);
    }
/* do the difference spectrum if IMODE=3, set by F3 key */
  if (imode == 3 )
   {
     if (bkgr == 1)
       {
         cmpfft (raw_buf, spc_buf, limit, pi);
/*          normal (raw_buf,spoints);               */
         for (index=1; index < limit/2; index++)
          spc_bak[index-1] = raw_buf[index];
       }
     else
       {
         cmpfft (raw_buf, spc_buf, limit, pi);
/*          normal (raw_buf, spoints);            */
         for (index = istps; index <= iendp; index++)
          raw_buf[index] = raw_buf[index] - spc_bak[index-1];
       }
    }

  /* find the interferogram error code */
    ercod = errcod (raw_data, limit, burst, lastpeak);
    lastpeak = burst;

  /* find the peak to peak value if imode =1 */
    if (imode == 0)
      {
       max_val = 0;
```

```
        min_val = 0;
        for (index = 1; index <= limit; index++)
         {
          max_val = max( raw_data[index], max_val);
          min_val = min( raw_data[index], min_val);
          }
        pktopk = max_val - min_val;
       }
      if (imode == 1 || imode == 3 )
        {
         minx_val = gh.sample_freq * (istps - 1);
         maxx_val = gh.sample_freq * iendp;
         maxy_val = 0.0;
         miny_val = 0.0;
         for (index = istps; index <= iendp; index++)
           {
           maxy_val = max (raw_buf[index], maxy_val);
           miny_val = min (raw_buf[index], miny_val);
           }
        }

   /* Plot the interferogram/spectral data to the screen */
   loop=loop ^ 1;
   _setactivepage(loop);
   _clearscreen (_GCLEARSCREEN);
   _setvieworg (0,0);
   logoega (2,12);
   _setvieworg (64,175);
   draw_axis (scan,imode);                /* draw axis */
   if (imode == 0)
      {
      plotr (raw_buf, istps, iendp, imode); /* display interferogram */
      _settextposition ( 3, 2);
      sprintf (buffer,"%5d",max_val);
      _outtext(buffer);
      _settextposition ( 23, 2);
      sprintf (buffer,"%5d",min_val);
      _outtext (buffer);
      _settextposition ( 24, 10);
      sprintf (buffer,"%5d",istps);
      _outtext (buffer);
      _settextposition (24, 70);
      sprintf (buffer,"%5d",iendp);
      _outtext (buffer);
      }
   if (imode == 1 || imode == 3 )
      {
      plotr (raw_buf, istps, iendp, imode); /* display spectrum */
      _settextposition ( 3, 1);
      sprintf (buffer,"%6.0f",maxy_val);
      _outtext (buffer);
      _settextposition ( 23, 1);
```

```
    sprintf (buffer,"%6.0f",miny_val);
    _outtext (buffer);
    _settextposition ( 25, 5);
    sprintf (buffer,"%6.0f",minx_val);
    _outtext (buffer);
    _settextposition ( 25, 70);
    sprintf (buffer,"%6.0f",maxx_val);
    _outtext (buffer);
    }
 if (imode == 3)                            /* reset the display code */
    bkgr=0;

 if (scan !=0)   /* display the interferogram error code */
   {
    if (imode == 0)
    {
    _settextposition (2, 35);
    _outtext ("peak-to-peak = ");
    _settextposition (2, 51);
    sprintf (buffer, "%051d", pktopk);
    _outtext (buffer);
    }
    _settextposition (2, 60);
    _outtext ("error code =");
    _settextposition (2,73);
    sprintf (buffer, "%01d", ercod);
    _outtext (buffer);
   }
    _settextposition (25,13);
    _outtext ("filename = ");
    _settextposition (25,25);
    _outtext (argv[1]);

    if (ch >= FINT && ch < FLEFT)
      jndex = (int) ch - 58;
    _settextposition (1,2);
    _outtext ("F");
    sprintf (buffer," %1d", jndex);
    _outtext (buffer);

  _setvisualpage(loop);
  }

  _setvideomode (_DEFAULTMODE);

}
/************************ end of program REPLAY ********************/
/************************function cmpfft ****************************/
/* CMPFFT

   This routine will Fourier transform an interferogram.  The program
   will rotate the interferogram and transform.  No phase correction
```

```
   or apodization is done.  This routine is to be only used for
   real-time display where phase and apodization functions are not
   absolutely required.  Do not use this routine for data analysis.

   routines called:
     rotate  - rotates an interferogram buffer
     burst   - finds the centerburst of an interferogram
     rfft    - calculates the Fourier transformation

-------------------------------------------------------------------*/
void cmpfft (raw_buf, spc_buf, ipoints, pi)
/*  The following global parameters are:
    raw_buf    - a work array used for transformation
    spc_buf    - an array containing the complex values of the transformation
    ipoints    - number of points in interferogram array
    pi         - value of pi
*/
float raw_buf[], spc_buf[], pi;
int ipoints;
{
/*  The following local parameters are:
    i,j,index  - indexing variables
    burst      - value containing the index of the interferogram centerburst
*/
  void rfft(),rotate();
  int fburst();
  int i, j, index, burst;

   for (i=1; i <= ipoints; i++)
    spc_buf[i] = raw_buf[i];

/* find the center burst of the interferogram */
/*  printf ("to burst\n");  */
/*  printf ("raw_buf[50]= %10.5f\n",raw_buf[50]); */
  burst=fburst(spc_buf,ipoints);
/*  printf ("after burst\n"); */

/* rotate the interferogram for the FFT */
/*  printf ("to rotate\n");  */
  rotate(burst, spc_buf, raw_buf, ipoints);
/*  printf ("after rotate\n");  */

/* Fourier transform the interferogram */
/*  printf ("to rfft\n");  */
  for (i=1, j=1; j <= ipoints; i+=2, j++)
    {
      spc_buf[i] = raw_buf[j];
      spc_buf[i+1] = 0.0;
/*      printf ("spc_buf[%04d]=%10.5f\n",i,spc_buf[i]);*/
/*      printf ("spc_buf[%04d]=%10.5f\n",i+1,spc_buf[i+1]);  */
    }
```

```
  rfft(spc_buf, ipoints, pi);
/*  printf ("after rfft\n"); */

/* compute the power spectrum */
/*  printf ("to power spectrum calculation\n"); */
   for ( i=1, j=0 ; i <= ipoints ; i+=2, j++)
   {
   raw_buf[j]= sqrt(spc_buf[i]*spc_buf[i]+spc_buf[i+1]*spc_buf[i+1]);
/*   printf ("raw_buf[%04d]=%10.5f\n",j,raw_buf[j]); */
   }
/*   printf ("after power spectrum calculation\n"); */
  }
/*********************** end of CMPFFT ******************************/
/*********************** function rfft ******************************/
/* RFFT

  This routine will compute the Fourier transform using the method
originally written by N. Brenner of Lincoln Laboratories

  routines called:
    NONE

------------------------------------------------------------------*/
void rfft (spc_buf, ipoints, pi)
/*  The following global parameters are:
    spc_buf  - the interferogram values stored in complex form
    ipoints  - number of points in interferogram
    pi       - value of pi
*/
float spc_buf[], pi;
int ipoints;
  {
    int i, n, istep, j, mmax, m;
    float wsin, theta, tempr, tempi, wr, wi, wtemp, wpr, wpi;

    n= 2 * ipoints;
    j=1;
   /* bit reversal section */
    for (i=1; i <= n ; i+=2)
     {
       if (j > i)
        {
/* Note: several statements have been commented out for the case
         where input imaginary values are always zero.  If this is
         not true, then these statements must be used.
*/
          tempr = spc_buf[j];
/*          tempi = spc_buf[j+1];   */
          spc_buf[j] = spc_buf[i];
/*          spc_buf[j+1] = spc_buf[i+1];  */
          spc_buf[i] = tempr;
/*          spc_buf[i+1] = tempi;  */
```

```
          }
        m=n/2;
        while ( m >= 2 && j > m )
          {
            j=j-m;
            m=m/2;
          }
        j=j+m;
      }

  /* compute the butterflies */

      mmax=2;
      while ( n > mmax )
        {
        istep= 2 * mmax;
        theta = 2.0 * pi /(float)mmax;
        wsin = sin(0.5 * theta);
        wpr = -2.0*wsin*wsin;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;
        for (m=1; m <= mmax; m+=2)
          {
            for (i=m; i <= n; i=i+istep)
              {
                j=i+mmax;
                tempr = wr*spc_buf[j] - wi*spc_buf[j+1];
                tempi = wr*spc_buf[j+1] + wi*spc_buf[j];
                spc_buf[j] = spc_buf[i] - tempr;
                spc_buf[j+1] = spc_buf[i+1] - tempi;
                spc_buf[i] = spc_buf[i] + tempr;
                spc_buf[i+1] = spc_buf[i+1] + tempi;
              }
            wtemp = wr;
            wr = wr*wpr - wi*wpi + wr;
            wi = wi*wpr + wtemp*wpi + wi;
          }
          mmax=istep;
        }
  }
/************************ end of RFFT ***********************************/
/************************ function fburst ******************************/
/* FBURST

    This routine will find the center burst of an interferogram array.
    The routine is a function call as the burst value is returned.

    routines called:
       NONE
--------------------------------------------------------------------*/
int fburst(raw_buf,ipoints)
```

```
/* The following global parameters are:
      raw_buf   - the interferogram array
      ipoints   - the number of points in interferogram
*/
float raw_buf[];
int ipoints;
 {
   int i,max_loc, min_loc;
   float max_val=0.0, min_val=0.0;

/*   printf ("ipoints in fburst= %04d\n",ipoints);
   printf ("raw_buf[50]= %10.5f\n",raw_buf[50]);*/
   for (i=1;i <= ipoints; i++)
    if (raw_buf[i] > max_val)
      {
        max_val=raw_buf[i];
        max_loc = i;
/*         printf("max_loc= %04d\n",max_loc);  */
      }
    else if (raw_buf[i] < min_val)
      {
        min_val = raw_buf[i];
        min_loc = i;
/*         printf ("min_loc= %04d\n",min_loc);  */
      }

     if (fabs((double) min_val) > max_val)
        return (min_loc);
   else
      return (max_loc);
  }
/************************ end of FBURST *******************************/
/************************ function normal *****************************/
/* NORMAL

   This routine is used to normalize the spectral buffer.

   routines called:
      NONE

-----------------------------------------------------------------*/
void normal (buffer, ipoints)
float buffer[];
{

  int index;
  float ssq = 0.0;

  for (index = 0; index < ipoints; index++)
    ssq += buffer[index] * buffer[index];

  if (ssq > 0.0)
```

```
    ssq = ipoints / sqrt (ssq);
  else
    ssq = 1.0;

  for (index = 0; index < ipoints; index++)
    buffer[index] *= ssq;
}
/*********************** end of normal ******************************/
/*********************** function rotate ****************************/
/* ROTATE

   This routine will rotate an interferogram buffer.  The buffer will
   be rotated so that the center burst is in array position 1.

   routines called:
      NONE

-------------------------------------------------------------------*/
void rotate (burst, raw_buf, spc_buf, ipoints)
/* The following parameters are:
    raw_buf  -  the input interferogram buffer
    spc_buf  -  the rotated interferogram buffer
    burst    -  the interferogram center burst array position
    ipoints  -  number of interferogram points is arrays
*/
float raw_buf[], spc_buf[];
int ipoints, burst;
 {
  int oindex, nindex;

  for (oindex=burst, nindex=1; oindex <= ipoints; oindex++, nindex++)
    spc_buf[nindex] = raw_buf[oindex];
/*  nindex-=1;      */
  for (oindex=1; oindex < burst; oindex++)
    {
      spc_buf[nindex] = raw_buf[oindex];
      nindex++;
     }
  }
/*********************** end of ROTATE ******************************/
/*********************** function draw_axis *************************/
/* DRAW_AXIS

   This routine will draw the axis for either an interferogram or
   spectrum display.

   routines called:
     Microsoft C graphics display routines

-------------------------------------------------------------------*/
void draw_axis (scan,imode)
/* The following parameters are:
```

```
     scan  - the scan number
     imode - display mode type; 0=interferogram, 1=spectrum
*/
int scan, imode;
{
  int  i, ih;
  char buffer[80];

  if (imode == 1)
    ih = 150;
  else
    ih = 0;

  _moveto (0, ih+0);    /* Print the X axis */
  _lineto (512, ih+0);
  _moveto (0,150);    /* Print the Y axis */
  _lineto (0,-150);

  for(i = 0; i <= 512; i += 64)  /*Print the X axis tick marks */
    {
    _moveto(i, ih+5);
    _lineto(i, ih+0);
    }

  for(i = 0; i <= 512; i += 32)
    {
    _moveto(i, ih+3);
    _lineto(i, ih+0);
    }

  for(i = 0; i <= 512; i += 16)
    {
    _moveto(i, ih+2);
    _lineto(i, ih+0);
    }

/*  for(i = 150; i > -150; i -= 25)   Print the Y axis tick marks
    {
    _moveto(-4, i+1);
    _lineto(0, i+1);
    }  */

   /* Label the axis */
   _settextposition(25,51);          /*  X AXIS */
   _outtext (" SCAN # ");
    sprintf(buffer,"%05d",scan);
   _settextposition(25,60);
   _outtext (buffer);
   if (imode == 3)  /* if difference spectrum add units */
     {
       _settextposition (25, 8);
       _outtext ("700");
```

```
      _settextposition (25,70);
      _outtext ("1400");
    }

  _settextposition(9,5);             /* Y AXIS */
  _outtext ("A");
  _settextposition(10,5);
  _outtext ("/");
  _settextposition(11,5);
  _outtext ("D");
  _settextposition(13,5);
  _outtext ("u");
  _settextposition(14,5);
  _outtext ("n");
  _settextposition(15,5);
  _outtext ("i");
  _settextposition(16,5);
  _outtext ("t");
  _settextposition(17,5);
  _outtext ("s");

}
/**************************** end of DRAW_AXIS ******************************/
/**************************** function logoega ******************************/
/*  logoega is a function used to create the CBDA logo for EGA graphics.
    The funtion requires two parameters, the x and y coordinates for the
    first letter "C". If the logo coordinates are outside the exceptable
    range, no logo will be plotted.

   author: John Ditillo
   modified by: Bob Kroutil

           logoega was based on the "old" CRDEC logo routine
           written by John T. Ditillo

   date: October 1992  */

void logoega(y,x)
int y, x;

{
  int xp, yp;

  if (y<23 & y>1 & x<76 & x>2)
    {

    /* draw the logo */
    _settextposition(y,x);
    _outtext ("C");

    _settextposition(y+1,x-1);
    _outtext ("B D");
```

```
    _settextposition(y+2,x);
    _outtext ("A");

    /* Calculate first pixel location */
    yp = y * 14 - 16;
    xp = x * 8 - 5;

    /* first benzene */
    _moveto(xp,yp);
    _lineto(xp-8,yp+3);
    _lineto(xp-8,yp+13);
    _lineto(xp,yp+17);
    _lineto(xp+8,yp+13);
    _lineto(xp+8,yp+3);
    _lineto(xp,yp);

    /* second benzene */
    _moveto(xp-8,yp+13);
    _lineto(xp-16,yp+17);
    _lineto(xp-16,yp+27);
    _lineto(xp-8,yp+31);
    _lineto(xp,yp+27);
    _lineto(xp,yp+17);

    /* third benzene */
    _moveto(xp+8,yp+13);
    _lineto(xp+16,yp+17);
    _lineto(xp+16,yp+27);
    _lineto(xp+8,yp+31);
    _lineto(xp,yp+27);

    /* fourth benzene */
    _moveto(xp-8,yp+31);
    _lineto(xp-8,yp+42);
    _lineto(xp,yp+45);
    _lineto(xp+8,yp+42);
    _lineto(xp+8,yp+31);

  }

}
/***********************  end of LOGOEGA ****************************/
/***********************  function plotr ****************************/
/* PLOTR

   This routine is used to scale and display the interferogram or
   spectrum.

   routines called:
    Microsoft C graphics routines

-----------------------------------------------------------------------*/
```

```
void plotr (buf,istps,iendp,imode)
/* The following parameters are:
      buf    - the array buffer to plot
      istps  - the starting point number to plot
      iendp  - the ending point number to plot
      imode  - the display mode (0=interferogram, 1=spectrum)
*/
float buf[];
int istps,iendp,imode;
{
  int index, x, y, ih, ip;
  float max, xscale, yscale;

  /* find the number of points to plot */
  ip = iendp - istps;

  /* find the largest value */
  for (index=istps, max=0.0; index < iendp; index++)
    {
    if ((fabs((double)buf[index])) > max)
      max = (float) (fabs((double)buf[index]));
    }

  /* Calculate the scaling factor */
  xscale = 512.0/ip;
  if (imode == 1)
    {
    yscale = 300.0/max;
    ih = 150;
    }
  else
    {
    yscale = 150.0/max;
    ih = 0;
    }

  /* plot the data */
  _moveto (0, (int) -(buf[istps] * yscale - ih));
  for (index=1; index < ip; index++)
    {
     x = (int) index * xscale;
     y = (int) -(buf[index+istps] * yscale - ih);
     _lineto (x,y);
    }

}
/*************************** end of PLOTR ****************************/
/*************************** function getspc *************************/
/* GETSPC

  This routine will get up to 4 black body spectra on the disk and
  read them into an array.  The stored spectra are SpectraCalc
```

```
  floating point binary format (FSP format - use the input and
  output commands in SpectraCalc).

  routines called:
   NONE

-------------------------------------------------------------------*/
void getspc (spc_bak, ipts, ch)
/* The following parameters are:
 spc_bak - the array that contains the 4 stored spectral responses
 ipts    - the number of points in the array
 ch      - a flag to tell which spectrum file to read
*/

float spc_bak[];
int ipts;
char ch;
{
  int fp3;
  float numpts, firstx, lastx, xunits, yunits, res;
  char afile[20];

/* load the black body spectra */

     if (ch == FSEL5)
       strcpy (afile,"f5.fsp");
     if (ch == FSEL6)
       strcpy (afile,"f6.fsp");
     if (ch == FSEL7)
       strcpy (afile,"f7.fsp");
     if (ch == FSEL8)
       strcpy (afile,"f8.fsp");

     if ((fp3 = open (afile,O_RDONLY|O_BINARY)) >= 0)
       {
        read (fp3, (char *) &numpts, 4);
        read (fp3, (char *) &firstx, 4);
        read (fp3, (char *) &lastx, 4);
        read (fp3, (char *) &xunits, 4);
        read (fp3, (char *) &yunits, 4);
        read (fp3, (char *) &res, 4);

        if ( read (fp3, spc_bak, 4 * ipts) != 4 * ipts)
          printf("\nUnable to read disk stored black body file.\n");

        close (fp3);
        }
   }
/**************************** end of getspc ***************************/
/**************************** function errcod *************************/
/* ERRCOD
```

```
  This routine will determine if the interferogram has an error.

  routines called:
     NONE

------------------------------------------------------------------*/
int errcod (raw_data, ipoints, burst, lastpeak)
/* The following parameters are:
   raw_buf    - the integer buffer array to test
   ipoints    - number of points in array
   burst      - the array location of the center burst
   lastpeak   - the last array location holding the previous center burst
*/

int raw_data[], burst, lastpeak, ipoints;
  {
    int ercod;

    ercod = 0;

    if (ipoints < 1024)
      ercod = 1;
    if (raw_data[burst] >= 32767)
      ercod = 2;
    if (lastpeak != burst)
      ercod = 3;
    if (burst > 500)
      ercod = 4;
    if (abs(raw_data[burst]) < 4096)
      ercod = 5;

/* NOTE: error code for bit toggle not yet implemented */

    return (ercod);
    }
```

Blank

APPENDIX D

MIDCOL DATA COLLECTION PROGRAM

```
/************************ program MIDCOL ********************************/
/*
 program MIDCOL                              Version 3.0

   This program is used to read interferogram data, display,
   interferogram data, and Fourier transform the data  for
   display.  This program will be used for data collection
   for the Midac interferometer.

   author: Bob Kroutil, Mike Housky

   date: August 1992

   routines called:
      plotr      - plots an interferogram or spectrum
      logoega    - prints the CRDEC logo
      draw_axis - draws the axis for the plots for either interferogram
                   or spectra
      cmpfft     - computes the fast Fourier transform
      normal     - normalizes the spectrum
      MidAqInit - initialize the Midac interferometer
      MidAqStartScan - set up scanning for Midac
      Microsoft C graphics routines

This software is property of the U.S. Army.  The distribution of this
code is unlimited.  This software can not be sold.  The U.S. Army is
not responsible for the results obtained through the use of this
software.
-----------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <graph.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include "headers.def"

#include <stddef.h>/* Standard ANSI headers*/
#include <conio.h>/* MSC-specific headers*/
#include <dos.h>

#include "middef.h"/* Midac-specific headers*/

/* ------------------------------------------------------------------ */
/*                Local definitions:                                  */
/* ------------------------------------------------------------------ */

/* MSC7/MSC6 Portability:                                             */

#ifdef MSC_VER
```

```
#if MSC_VER >= 700
#define outp _outp
#define inp  _inp
#endif
#endif

#define TIMEOUT 30.0            /* DMA Completion timeout, in seconds   */

/* Defaults for MidAqInit:                                               */

#define DMA                1    /* Default DMA channel                  */
#define DMAPAGE            0x83 /* DMA page register port for default   */
                 /*   channel                              */
#define IRQ                2    /* Default IRQ channel                  */
#define GAIN               0    /* Default signal gain level (0-7)      */
#define BUFPTS             16384 /* Default DMA buffer size in data     */
                 /*   points                               */
#define MAXDMA             0xFF80 /* Maximum DMA buffer size in bytes   */

            /* Note: MAXDMA must be less than the "ideal" limit of  */
            /* 64K for the GetDmaBuffer functicn to work properly.  */

/*
            System board (PC/AT) I/O definitions:
*/

#define SYS_DMA1           0x00   /* Base of byte DMA controller        */

/* These ports are channel-independent:                                  */

#define DMA_STAT  (SYS_DMA1+ 8) /* (R) Status register                  */
#define DMA_CMD   (SYS_DMA1+ 8) /* (W) Command register                 */
#define DMA_REQ   (SYS_DMA1+ 9) /* (W) Request register                 */
#define DMA_WSMR  (SYS_DMA1+10) /* (W) Write single mask register       */
#define DMA_MODE  (SYS_DMA1+11) /* (W) Mode register                    */
#define DMA_CLRF  (SYS_DMA1+12) /* (W) Clear byte pointer flip-flop     */
#define DMA_TEMP  (SYS_DMA1+13) /* (R) Temporary register               */
#define DMA_MCLR  (SYS_DMA1+13) /* (W) Master Clear                     */
#define DMA_CMSK  (SYS_DMA1+14) /* (W) Clear mask register              */
#define DMA_WAMR  (SYS_DMA1+15) /* (W) Write all mask register bits     */

/* These occur 4 times, once for each channel. Add 2*(channel number)   */
/* to get true port address:                                             */

#define DMA_ADDR (SYS_DMA1+ 0)  /* (R/W) Base or current address        */
#define DMA_CTR  (SYS_DMA1+ 1)  /* (R/W) Base or current word count     */

#define SYS_PIC1           0x20  /* Base of primary interrupt controller */
#define PIC1_CMD  (SYS_PIC1+0)  /* (W) Command register (OCW2/OCW3)     */
#define PIC1_STAT (SYS_PIC1+0)  /* (R) Status register (ISR or IRR)     */
#define PIC1_MASK (SYS_PIC1+1)  /* (R/W) Interrupt mask register        */
```

```
#define SYS_PIC2        0xA0    /* Base of secondary int. controller    */
#define PIC2_CMD  (SYS_PIC2+0)  /* (W) Command register (OCW2/OCW3)     */
#define PIC2_STAT (SYS_PIC2+0)  /* (R) Status register (ISR or IRR)     */
#define PIC2_MASK (SYS_PIC2+1)  /* (R/W) Interrupt mask register        */

#define PICC_EOI        0x20    /* OCW2 (nonspecific) End-Of-Interrupt  */
                /*   command                            */

/*
            Local Macros:
*/

#define PtrToLong(p) (((long)FP_SEG(p) << 4) + (long)FP_OFF(p))
                /* Macro to convert far pointer to      */
                /*   20-bit absolute address            */

#define DisableDma(ch) outp(DMA_WSMR, (ch)+4)   /* Disable DMA channel  */
#define EnableDma(ch)  outp(DMA_WSMR, (ch))     /* Enable DMA channel   */

/* Input and output from read-only command port, a shadow copy of the   */
/* port value is kept in MidGbl.CmpPort:                                */

#define CmdIn()    (MidGbl.CmdPort)
#define CmdOut(val) (outp(MID_CMD, MidGbl.CmdPort = (int)(val)), \
                    outp(MID_CMD, MidGbl.CmdPort))

/* ------------------------------------------------------------------- */
/*              Global variables:                                      */
/* ------------------------------------------------------------------- */

MidAqGlobalType near MidGbl;    /* Global paramater/context variables   */

static int near DmaPageTable[8] = /* Table of DMA page register ports   */
            { 0x87, 0x83, 0x81, 0x82, -1, 0x8B, 0x89, 0x8A };

/* The following global parameters are the following:
      LIMIT    = the number of interferogram points
      PLIMIT   = the number of sampled midac interferogram points
      SLIMIT   = the number of spectral points
      GH_LIMIT = the number of bytes in the global interferogram header
      SH_LIMIT = the number of points in the subfile interferogram header
      FEND     = the key code to exit the program
      FRIGHT   = the key code to expand the interferogram display
      FHOME    = the key code to reset the interferogram display
      FLEFT    = the key code to compress the interferogram display
      FINT     = the key code to display interferograms
      FSPEC    = the key code to display spectra
      FSCOL    = the key code to collect interferograms to disk
      FDIFF    = the key code to display a difference spectrum
      FBACK    = the key code to calculate a background spectrum
      FSEL5    = the key code to subtract the disk file f5.fsp
      FSEL6    = the key code to subtract the disk file f6.fsp
```

```
      FSEL7     = the key code to subtract the disk file f7.fsp
      FSEL8     = the key code to subtract the disk file f8.fsp
      ROLLL     = the key code to roll the display data to the left
      ROLLR     = the key code to roll the display data to the right
      PMODE     = the file read/write attributes
*/

#define LIMIT 1024
#define PLIMIT 1024
#define SLIMIT 513
#define GH_LENGTH 512
#define SH_LENGTH 64
#define FEND 79
#define FRIGHT 68
#define FHOME 71
#define FLEFT 67
#define FINT 59
#define FSEL5 63
#define FSEL6 64
#define FSEL7 65
#define FSEL8 66
#define FSPEC 60
#define FSCOL 61
#define FDIFF 62
#define ROLLL 75
#define ROLLR 77
#define PMODE 0644

main(argc, argv)
int argc;
char *argv[];
{
/* The following parameters are:
    raw_buf          - the interferogram buffer (real values)
    spc_buf          - the complex interferogram buffer, also used as a
                       work array
    pi               - value of the constant pi
    scan             - the scan number
    index            - an indexing variable
    fp2              - file open variables
    lpoints          - number of points in interferogram to display
    spoints          - number of points in the spectrum
    imode            - 0=display interferogram, 1=display spectrum
    inode            - set data collect switch
    loop             - graphics display page
    wscan            - scan number writing to disk
    ch               - used for an input
    bkgr             - the collect background flag
    ispts            - starting spectral plotting point for difference
                       spectrum
    iendp            - ending spectral plotting point for difference
                       spectrum
```

```
    lastpeak          - last array position of interferogram center burst
    extp              - the input extension filename
    drivep            - the input drive filename
    dirp              - the input directory filename
    icount2           - the index for number of bytes to copy
    outname           - the global header filename
    dirname           - the input filename to store to disk
    idate             - the array to hold the date
    itime             - the array to hold the time
    res               - the instrument resolution
    coll              - data collection mode
    itype             - integer data type
    speed             - interferometer scan speed
    mirror            - interferometer mirror movement
    sample            - spectral wavenumber sampling interval
    startf            - starting wavenumber
    stopf             - ending wavenumber
    mxwav             - maximum wavenumber frequency that can be sampled
    zcross            - number of zero crossings per sampled point
    temp              - ambient temperature
    barp              - barometric pressure
    humid             - relative humidity
    wind              - wind speed
    windd             - wind direction
    sendir            - sensor pointing direction
    precc             - precipitation code
    sensid            - array for sensor name
    opernam           - array for operators name
    global_header,gh  - the global header structure
    scan_header,sh    - the subfile header structure
    igain             - the A/D gain of the interferometer (0 - 7)
*/
  int MidAqInit(), MidAqSetGain();
  void MidAqStartScan();
  int wrtint();
  void dispint(), dispspec(), diffspc(), logoega(), getspc();
  float raw_buf[LIMIT+1], spc_buf[2*LIMIT+2], spc_bak[LIMIT+1], pi;
  int inode, wscan, bkgr, spoints, jndex = 1;
  int scan, index, imode, loop=0, lastpeak, istps, iendp, ichng;
  char ch, buffer[4], outname[10], dirname[40], idate[10], itime[10];
  double res, mirror, speed, sample, startf, stopf, barp, mxwav;
  char comm1[64], comm2[64], comm3[64], comm4[64];
  int coll, itype, temp, humid, wind, windd, sendir, precc, ierr;
  int fp2, burst, zcross, maxscan, igain;
  char sensid[20], opernam[10], extp[4], drivep[10], dirp[10];
  size_t hdcl, icount2=20;
  unsigned long t0,t1;
  struct global_header gh;
  struct scan_header sh;

/* set the maximum number of scans to collect by an input switch */
   if (argc == 2)
```

```
      maxscan = 550;
    else
       maxscan = 3000;

/* ask the user to input an output data collection filename */
    _clearscreen (_GCLEARSCREEN);
    printf("\nMIDCOL  -    Midac remote sensing data collection program
Version 3.0\n");
    printf("\nThe program switch is set to collect up to %d interferograms to
disk.\n",maxscan);
    printf("\nInput the data filename to store to disk: ");
    scanf ("%s",dirname);
    hdcl = 10;
    memset (&outname,32,hdcl);
    _splitpath (dirname, drivep, dirp, outname, extp);
    strupr (outname);

/* initialize the input buffers */
    hdcl =64;
    memset(&comm1,32,hdcl);
    memset(&comm2,32,hdcl);
    memset(&comm3,32,hdcl);
    memset(&comm4,32,hdcl);
    printf ("\nInput four lines for comments:\n");
    gets (comm1);
    printf (">>");
    gets (comm1);
    printf (">>");
    gets (comm2);
    printf (">>");
    gets (comm3);
    printf (">>");
    gets (comm4);

 /* set up the graphics mode and clear screen */
   _setvideomode (_ERESCOLOR);
   _setbkcolor (_BLUE);
   _settextposition (13, 20);
   _outtext ("Please Wait -- Initializing Interferometer");

/*==============create the global header================================*/
  /* create a new global header */

/* clear the global header buffers with blanks */
  hdcl=512;
  memset (&gh,32,hdcl);
  hdcl = 64;

/* initialize the default global header data parameters */
  coll = 0;                 /* data collection mode */
  itype = 1;                /* integer data type */
  res = 8.0;                /* instrument resolution */
```

```
   speed = 2.2;              /* interferometer scan speed */
   mirror = 2.5;             /* interferometer mirror velocity */
   sample = 3.856933;        /* sampling frequency parameter */
   startf = 0.0;             /* starting wavenumber */
   stopf = 1974.75;          /* ending wavenumber */
   mxwav = 15796.0;          /* maximum sampling frequency */
   zcross = 800;             /* number of laser fringes per point * 100 */
   temp = 0;                 /* ambient temperature */
   barp = 0.0;               /* barometric pressure */
   humid = 0;                /* relative humidity */
   wind = 0;                 /* wind speed */
   windd = 0;                /* wind direction */
   sendir = 0;               /* sensor direction */
   precc = 0;                /* precipitation code */
   strcpy (sensid,"Midac unit #120     "); /* set the sensor name */
   strcpy (opernam,"          ");      /* blank out the operators name */

/*---------------------------------------------------------------*/

/* stuff in the integer and double header information into
   the correct locations */

   gh.collect_mode = coll;
   gh.integer_type = itype;
   gh.scan_size = LIMIT;
   gh.resolution = res;
   gh.scan_speed = speed;
   gh.mirror_velocity = mirror;
   gh.sample_freq = sample;
   gh.start_freq = startf;
   gh.stop_freq = stopf;
   gh.max_wav = mxwav;
   gh.zercross = zcross;
   gh.ambient_temp = temp;
   gh.bar_pressure = barp;
   gh.humidity = humid;
   gh.wind_speed = wind;
   gh.wind_direction = windd;
   gh.sensor_direction = sendir;
   gh.precip_code = precc;

/* copy the sensor id */
   icount2 = 20;
   memcpy (&gh.sensor_id, &sensid, icount2);

/* copy the comment field */
   icount2=64;
   memcpy (&gh.comm1, &comm1, icount2);
   memcpy (&gh.comm2, &comm2, icount2);
   memcpy (&gh.comm3, &comm3, icount2);
   memcpy (&gh.comm4, &comm4, icount2);
```

```
/* find the starting date and time */
  _strtime (itime);
  icount2 = 10;
  memcpy (&gh.start_time, &itime, icount2);
  _strdate (idate);
  memcpy (&gh.date, &idate, icount2);

/* input the operators name */
  memcpy (&gh.operator, &opernam, icount2);

/* input the filename into the header */
  memcpy (&gh.filename, &outname, icount2);

  if (fp2 = creat (dirname, PMODE) < 0 )
    {
     _setvideomode (_DEFAULTMODE);
     printf ("\n\"MIDCOL\" is unable to create %s\n",dirname);
     exit(2);
    }
  if ((fp2 = open (dirname, O_WRONLY|O_BINARY)) < 0)
     {
      _setvideomode (_DEFAULTMODE);
      printf ("\n\"MIDCOL\" is unable to open %s\n",dirname);
      exit(2);
     }
  /* write the global header information */
    write (fp2, &gh, GH_LENGTH);

  /* set the parameter values for data collection */
  pi=4.*atan(1.);  /* the value of pi */
  imode = 0;       /* 0=display interferogram ; 1=display spectrum */
  istps = 1;       /* the starting point to display */
  iendp = 400;     /* the ending point to display */
  ichng = 50;      /* the display number of points to roll screen */
  spoints = LIMIT; /* set the maximum point number to roll screen */
  wscan = 1;       /* initialize number of scans written to disk */
  inode = 0;       /* determines status of disk file */
  bkgr = 1;        /* set the background flag to collect */
  scan = -1;       /* initialize the scan data collection value */
  igain = -1;      /* have the gain initialize to initial value */

  /* This is the main loop for data collection to proceed */

 /* initialize the interferometer with scanning parameters */
    index = MidAqInit( -1, -1, igain, PLIMIT);
    if (index)
    {
   _setvideomode (_DEFAULTMODE);
            printf("Error: MidAqInit returned %d\n", index);
            exit (2);
    }
/*    printf("MidCol initialized:\n");
```

```
    printf("  DMA Buffer at %Fp = %061X\n", MidGbl.DmaBuffer,
                 PtrToLong(MidGbl.DmaBuffer)); */

/************************************************************************/
/* check the scan rate and store value in the header buffer */
   t0 = (unsigned long)clock();
   MidAqStartScan();
   while (!MidGbl.DmaDone)
   {
    t1 = (unsigned long) clock();
    if ((t1-t0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
    {
     _setvideomode (_DEFAULTMODE);
     printf("===> ERROR - no signal from interferometer <===");
     exit (2);
    }
   }
   MidGbl.DmaActive = 0;
   speed = 1.0/((float)(t1-t0)/(float)CLOCKS_PER_SEC);
   mirror = 0.25 * speed;
   gh.scan_speed = speed;
   gh.mirror_velocity = mirror;
/************************************************************************/
/************************************************************************/
 /* check the instrument gain -- if too low, then increase gain
                              if too high, then decrease gain */
/*    igain++;
    MidAqStartScan();
    t0 = (unsigned long)clock();
    while (!MidGbl.DmaDone)
     {
      t1 = (unsigned long)clock();
      if ((t1-t0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
        {
         _setvideomode (_DEFAULTMODE);
         printf("Error: Timeout on DMA completion\n");
         exit (2);
        }
     }
    MidGbl.DmaActive = 0;
    for (index=0; index < LIMIT-1; index++)
        raw_buf[index+1] = (float) MidGbl.DmaBuffer[index];

    burst = fburst(raw_buf,LIMIT-1);
    while(fabs(raw_buf[burst]) <= 16384. && igain <= 7)
     {
      raw_buf[burst] *= 2.;
      igain +=1;
     }
      MidAqSetGain(igain);
      printf(".... setting the instrument A/D gain to = %d",igain);
      MidAqStartScan();
```

```
      t0 = (unsigned long)clock();
      while (!MidGbl.DmaDone)
        {
          t1 = (unsigned long)clock();
          if ((t1-t0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
            {
              _setvideomode (_DEFAULTMODE);
              printf("Error: Timeout on DMA completion\n");
              exit (2);
            }
        }
       MidGbl.DmaActive = 0; */
/***********************************************************************/

 /* loop to collect interferogram data */

tloop:
    scan++;

 /* read in the interferogram data from the interferometer */
  MidAqStartScan();
  t0 = (unsigned long) clock();
  while (!MidGbl.DmaDone)
  {
    t1 = (unsigned long) clock();
    if ((t1-t0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
    {
      _setvideomode (_DEFAULTMODE);
      printf("Error: Timeout on DMA completion\n");
      exit (2);
    }

/*------------------ select user mode -----------------------------*/
     if (kbhit() != 0)  /* check to see if a key was pressed */
       {
         ch=getch();
         if (ch == FEND)  /* exit program */
           {
             if (inode == 1) /* if writing to disk update global header */
              {
               lseek (fp2, 0L, 0); /* rewind the file to write header */
               gh.stop_scan = wscan - 1;  /* insert the number of scans in
header */
               _strtime (itime);    /* input the ending time into header */
               memcpy (&gh.stop_time, &itime, icount2);
               write (fp2, &gh, GH_LENGTH);/* write global header */
               close (fp2);
               }
            _setvideomode(_DEFAULTMODE);
            exit(1);
            }
         if (ch == FRIGHT) /* expand screen display */
```

```
            {
            iendp = iendp - ichng;
            istps = istps + ichng;
            if (istps >= iendp)
              {
               istps = istps - ichng;
               iendp = iendp + ichng;
              }
            }
          if (ch == FLEFT)/* contract the screen display */
            {
            iendp = iendp + ichng;
            istps = istps - ichng;
            if (istps < 1 ) istps = 1;
            if (iendp > spoints) iendp = spoints;
            }
          if (ch == ROLLR) /* roll the data to the right */
            {
            iendp = iendp - ichng;
            istps = istps - ichng;
            if (istps < 1 )
              {
               istps = 1;
               iendp = iendp + ichng;
              }
            }
          if (ch == ROLLL) /* roll the data to the left */
            {
            iendp = iendp + ichng;
            istps = istps + ichng;
            if (iendp > spoints)
                {
                 iendp = spoints;
                 istps = spoints - ichng;
                }
            }
           if (ch == FINT)/* display interferogram */
            {
             imode=0;
             istps = 1;
             iendp = 400;
             spoints = LIMIT;
/*             _setbkcolor (_BLUE);    */
            }
          if (ch == FSPEC)/*  display spectrum */
            {
             imode=1;
             istps = 1;
             iendp = 512;
             spoints= iendp;
/*             _setbkcolor (_BLUE);    */
            }
```

```
          if (ch == FSCOL) /* set disk data collection turned on */
           {
             imode = 2;
             inode = 1;
/*             _setbkcolor (_BLACK);  */
           }
          if (ch == FDIFF) /* display the difference spectrum */
           {
             imode = 3;
             bkgr = 1;
             istps = 181;
             iendp = 363;
             spoints = 512;
/*             _setbkcolor (_BLUE);   */
           }
           if (ch == FSEL5 || ch == FSEL6 || ch == FSEL7 || ch == FSEL8)
           {
            imode = 3;
            bkgr = 0;
            istps = 181;
            iendp = 363;
            spoints = 512;
            getspc (spc_bak, spoints, ch);
/*            _setbkcolor (_BLUE);   */
           }
           if (ch >= FINT && ch < FLEFT)
             jndex = (int) ch - 58;
      }
 /* return to check the keyboard if the scan is not finished */
    }
/*-------------------------------------------------------------------*/
    MidGbl.DmaActive = 0;

  /* convert the integer array to an ungain ranged floating array */
  for (index = 0; index < LIMIT; index++)
    raw_buf[index+1] = (float) MidGbl.DmaBuffer[index];

  raw_buf[0]=0.0;
  spc_buf[0]=0.0;

/* set up the graphics to plot */
  loop = loop ^ 1;
  _setactivepage(loop);
  _clearscreen(_GCLEARSCREEN);
  _setvieworg(0,0);
   logoega(2,12);
  _setvieworg(64,175);

/* do the correct math operation for each selection */
/* display the interferogram to the screen */
  if (imode == 0)
     dispint (raw_buf, istps, iendp, imode, scan, jndex);
```

```
/* display the spectrum to the screen */
  else if (imode == 1)
     dispspec (raw_buf, spc_buf, LIMIT, istps, iendp, pi, imode, scan, sample,
               jndex);
  else if (imode == 2)
     {
/* exit data collection if too many interferograms have been collected */
     if (wscan > maxscan)
       {
         lseek (fp2, OL, 0); /* rewind the file header */
         gh.stop_scan = wscan - 1; /* insert the number of scans in header */
         _strtime(itime); /* get the ending time to put into header */
         memcpy (&gh.stop_time, &itime, icount2);
         write (fp2, &gh, GH_LENGTH); /* write the global header */
         close(fp2);
         _setvideomode(_DEFAULTMODE);
         exit(2);
        }

/* write the interferogram to the disk */
     lastpeak=wrtint (raw_buf, LIMIT, wscan, lastpeak, outname, dirname,
                      fp2);
     wscan++;
     }
  else
/* write the difference spectrum to the screen */
     {
     diffspc (raw_buf, spc_buf, spc_bak, pi, bkgr, imode, istps,
             iendp, scan, LIMIT, sample, jndex);
             bkgr=0;
     }

/* loop to get more data */

  _setvisualpage(loop);
  goto tloop;

}
/************************ end of program MIDCOL *********************/
/************************function dispint ****************************/
/* DISPINT

  This routine will display the interferogram on the screen for the
  real-time data collect option

  routines called:
  draw_axis     -  draw an axis to the screen
  plotr         -  plot the interferogram on the screen

-----------------------------------------------------------------*/
void dispint (raw_buf, istps, iendp, imode, scan, jndex)
/* The following global parameters are :
```

```
   raw_buf   - the interferogram data points to display
   imode     - the plotting mode to display 0=interferogram display
   istps     - the starting point to display
   iendp     - the ending point to display
   scan      - the scan number of the interferogram
   jndex     - the menu number to display on the screen
*/
float raw_buf[];
int istps, iendp, imode, scan, jndex;

 {
  void draw_axis(), plotr();
  int i;
  long int max_val=0, min_val=0, pktopk;
  char buffer[5];

/* find the peak to peak value of the interferogram */
  for (i = istps; i < iendp; i++)
   {
    max_val = max (MidGbl.DmaBuffer[i],max_val);
    min_val = min (MidGbl.DmaBuffer[i],min_val);
   }
   pktopk = max_val - min_val;

/* plot the interferogram data to the screen */
  draw_axis (scan,imode);
  plotr (raw_buf, istps, iendp, imode);
  _settextposition ( 2, 54);
  _outtext ("peak-to-peak = ");
  _settextposition ( 2, 70);
  sprintf (buffer,"%5ld",pktopk);
  _outtext(buffer);
  _settextposition (3, 2);
  sprintf (buffer,"%5d",max_val);
  _outtext(buffer);
  _settextposition (23, 2);
  sprintf (buffer,"%5d",min_val);
  _outtext(buffer);
  _settextposition (24, 10);
  sprintf (buffer,"%5d",istps);
  _outtext(buffer);
  _settextposition (24,70);
  sprintf (buffer,"%5d",iendp);
  _outtext(buffer);
  _settextposition (1,2);
  _outtext("F");
  sprintf (buffer," %ld",jndex);
  _outtext (buffer);
 }
/*************************end of dispint ******************************/
/*************************function dispspec ***************************/
/* DISPSPEC
```

```
   This is the spectral display routine.  This routine will
Fourier transform and display each collected interferogram.

routines called:
cmpfft     -  Fourier transform
plotr      -  plot spectrum to screen
draw_axis  -  draw the axis to the screen

-----------------------------------------------------------------*/
void dispspec (raw_buf, spc_buf, limit, istps, iendp, pi, imode, scan, sample,
               jndex)
/* The following global variables are:
  raw_buf - the collected interferogram buffer
  spc_buf - the fourier transformed spectral buffer
  limit   - the number of points to transform
  istps   - the starting point to display
  iendp   - the ending point to display
  pi      - the value of PI
  imode   - the display mode; 1 = spectral buffer
  scan    - the scan number to display
  sample  - the sampling point spacing in wavenumbers
  jndex   - the menu option to display on the screen
*/
float raw_buf[], spc_buf[], pi, sample;
int istps, iendp, imode, scan, limit, jndex;
  {
   void cmpfft(), plotr(), draw_axis();
   float minx_val, maxx_val, miny_val = 0.0, maxy_val = 0.0;
   int i;
   char buffer[6];

/* do the fourier transform */
   cmpfft (raw_buf, spc_buf, limit, pi);

/* find the maximum and minimum values for the plotted spectrum */
   minx_val = sample * (istps-1);
   maxx_val = sample * iendp;
   for (i= istps; i < iendp; i++)
     maxy_val = max (raw_buf[i], maxy_val);

/* plot the spectrum data to the screen */
   draw_axis (scan,imode);
   plotr (raw_buf, istps, iendp, imode);
   _settextposition ( 3, 1);
   sprintf (buffer,"%6.0f",maxy_val);
   _outtext (buffer);
   _settextposition ( 23, 1);
   sprintf (buffer,"%6.0f",miny_val);
   _outtext (buffer);
   _settextposition ( 25, 5);
   sprintf (buffer,"%6.0f",minx_val);
   _outtext (buffer);
```

```
   _settextposition ( 25, 70);
   sprintf (buffer,"%6.0f",maxx_val);
   _outtext (buffer);
   _settextposition (1,2);
   _outtext("F");
   sprintf (buffer," %ld", jndex);
   _outtext (buffer);
   }
/************************end of dispspec ****************************/
/************************function wrtint *****************************/
/* WRTINT

   This routine will write an interferogram to the disk.

   routines called:
   errcod   -   find the interferogram error code
   fburst   -   find the interferogram centerburst

------------------------------------------------------------------*/
int wrtint (raw_buf, limit, wscan, lastpeak, outname, dirname, fp2)
/* The following global parameters are:
 raw_buf - the interferogram collected on the Midac
 limit   - the number of points in the array buffer
 wscan   - the last interferogram nummber written to disk
 lastpeak- the last interferogram burst position
 outname - the header name to store
 dirname - the directory name to store to disk
 fp2     - file pointers for disk I/O
*/
int wscan, limit, lastpeak, fp2;
float raw_buf[];
char dirname[], outname[];

  {
   int errcod(), fburst();
   int burst, ercod;
   size_t hdcl=64, icount2=10;
   char itime[10], buffer[4];
   struct scan_header sh;
   struct global_header gh;

/* initialize the subfile header information */
   memset (&sh, 32, hdcl);   /* initialize the subfile header buffer */
   burst = fburst (raw_buf, limit); /* find the center burst */
   if (wscan == 1)
    lastpeak = burst;
   sh.scan_number = wscan;          /* insert the scan number */
   sh.peak_location = burst;        /* centerburst position */
   sh.gain = MidGbl.GainVal;        /* interferogram A/D gain */
   sh.coadd = 1;      /* set the number of coadded interferograms */
   ercod = errcod (raw_buf, limit, burst, lastpeak);
   sh.error = ercod;                /* interferogram error code */
```

```
   lastpeak = burst;                  /* set the last peak position
                                         for the centerburst */

/* put the header name into the source filename field */
   memcpy (&sh.filename, &outname, icount2);

/* find the scan time to put into the header */
  _strtime (itime);
   memcpy (&sh.scan_time, &itime, icount2);

/* write the interferogram to disk */
   /* write the subfile header information */
   write (fp2, &sh, SH_LENGTH);
   /* write the interferogram data to disk */
   write (fp2, MidGbl.DmaBuffer, limit*2);

/* display the information the the screen */
  _settextposition( 12, 20);
  _outtext("COLLECTING INTERFEROGRAM DATA TO DISK");
  _settextposition( 14, 20);
  _outtext("filename = ");
  _settextposition( 14, 32);
  _outtext(dirname);
  _settextposition( 16, 20);
  _outtext("interferogram number = ");
  _settextposition( 16, 44);
  sprintf (buffer, "%04d", wscan);
  _outtext (buffer);
  _settextposition ( 18, 20);
  _outtext("error code = ");
  _settextposition ( 18, 33);
  sprintf (buffer, "%01d", ercod);
  _outtext (buffer);
  return(lastpeak);
  }
/************************end of wrtint ****************************/
/************************function diffspc ***************************/
/* DIFFSPC

  This routine will display a difference spectrum to the screen.

  routines called:
  cmpfft       -    Fourier transform
  normal       -    normalize a spectral buffer
  plotr        -    plot a spectral buffer to the screen
  draw_axis    -    plot the axis labels to the screen

---------------------------------------------------------------*/
void diffspc (raw_buf, spc_buf, spc_bak, pi, bkgr, imode, sstart,
              send, scan, limit, sample, jndex)
/* The following parameters are:
             raw_buf  -  real array of interferogram values
```

```
    spc_buf   -  real array of spectral values
    spc_bak   -  real array of spectral background values
    pi        -  the value of pi
    bkgr      -  the background computation switch
    imode     -  the data display mode
    sstart    -  the starting point to plot the difference spectrum
    send      -  the ending point to plot the difference spectrum
    limit     -  the interferogram array size
    sample    -  the sampling point increment (wavenumbers)
    jndex     -  the menu option number to display on the screen
*/

float raw_buf[], spc_buf[], spc_bak[], pi, sample;
int bkgr, imode, sstart, send, scan, limit, jndex;
  {
   void cmpfft(), normal(), plotr(), draw_axis();
   float minx_val, maxx_val, miny_val=0.0, maxy_val=0.0;
   int index;
   char buffer[6];

   if (bkgr == 1)
     {
      cmpfft (raw_buf, spc_buf, limit, pi);
/*     normal (raw_buf, spoints);   */
      for (index=1; index <= limit/2; index++)
        spc_bak[index-1] = raw_buf[index];
              }
   else
     {
      cmpfft (raw_buf, spc_buf, limit, pi);
/*     normal (raw_buf, spoints);  */
      for (index= sstart; index < send; index++)
       {
         raw_buf[index]=raw_buf[index]-spc_bak[index-1];
         miny_val = min (raw_buf[index], miny_val);
         maxy_val = max (raw_buf[index], maxy_val);
       }
      draw_axis( scan, imode);
      plotr (raw_buf, sstart, send, imode);

/* annotate the screen with the display ranges */
     minx_val = sample * (sstart-1);
     maxx_val = sample * send;
     _settextposition ( 3, 1);
     sprintf (buffer,"%6.0f",maxy_val);
     _outtext (buffer);
     _settextposition ( 23, 1);
     sprintf (buffer,"%6.0f",miny_val);
     _outtext (buffer);
     _settextposition ( 25, 5);
     sprintf (buffer,"%6.0f",minx_val);
     _outtext (buffer);
```

```
      _settextposition ( 25, 70);
      sprintf (buffer,"%6.0f",maxx_val);
      _outtext (buffer);
      _settextposition ( 1, 2);
      _outtext("F");
      sprintf (buffer," %ld", jndex);
      _outtext (buffer);
      }
   }
/*************************end of diffspc ****************************/
/************************function cmpfft ****************************/
/* CMPFFT

   This routine will Fourier transform an interferogram.  The program
   will rotate the interferogram and transform.  No phase correction
   or apodization is done.  This routine is to be only used for
   real-time display where phase and apodization functions are not
   absolutely required.  Do not use this routine for data analysis.

   routines called:
     rotate  - rotates an interferogram buffer
     burst   - finds the centerburst of an interferogram
     rfft    - calculates the Fourier transformation

--------------------------------------------------------------------*/
void cmpfft (raw_buf, spc_buf, ipoints, pi)
/*  The following global parameters are:
    raw_buf    - a work array used for transformation
    spc_buf    - an array containing the complex values of the transformation
    ipoints    - number of points in interferogram array
    pi         - value of pi
*/
float raw_buf[], spc_buf[], pi;
int ipoints;
{
/*  The following local parameters are:
    i,j,index  - indexing variables
    burst      - value containing the index of the interferogram centerburst
*/
  void rfft(),rotate();
  int fburst();
  int i, j, index, burst;

   for (i=1; i <= ipoints; i++)
    spc_buf[i] = raw_buf[i];

/* find the center burst of the interferogram */
/*  printf ("to burst\n");  */
/*  printf ("raw_buf[50]= %10.5f\n",raw_buf[50]); */
  burst=fburst(spc_buf,ipoints);
/*  printf ("after burst\n"); */
```

```
/* rotate the interferogram for the FFT */
/*  printf ("to rotate\n");  */
  rotate(burst, spc_buf, raw_buf, ipoints);
/*  printf ("after rotate\n");  */

/* Fourier transform the interferogram */
/*  printf ("to rfft\n");  */
  for (i=1, j=1; j <= ipoints; i+=2, j++)
    {
      spc_buf[i] = raw_buf[j];
      spc_buf[i+1] = 0.0;
/*      printf ("spc_buf[%04d]=%10.5f\n",i,spc_buf[i]);*/
/*      printf ("spc_buf[%04d]=%10.5f\n",i+1,spc_buf[i+1]);  */
    }

  rfft(spc_buf, ipoints, pi);
/*  printf ("after rfft\n"); */

/* compute the power spectrum */
/*  printf ("to power spectrum calculation\n"); */
   for ( i=1, j=0 ; i <= ipoints ; i+=2, j++)
   {
   raw_buf[j]= sqrt(spc_buf[i]*spc_buf[i]+spc_buf[i+1]*spc_buf[i+1]);
/*    printf ("raw_buf[%04d]=%10.5f\n",j,raw_buf[j]); */
   }
/*    printf ("after power spectrum calculation\n"); */
  }
/************************ end of CMPFFT ******************************/
/************************ function rfft *******************************/
/* RFFT

  This routine will compute the Fourier transform using the method
originally written by N. Brenner of Lincoln Laboratories

  routines called:
    NONE

-------------------------------------------------------------------*/
void rfft (spc_buf, ipoints, pi)
/*  The following global parameters are:
    spc_buf   - the interferogram values stored in complex form
    ipoints   - number of points in interferogram
    pi        - value of pi
*/
float spc_buf[], pi;
int ipoints;
  {
    int i, n, istep, j, mmax, m;
    float wsin, theta, tempr, tempi, wr, wi, wtemp, wpr, wpi;

    n= 2 * ipoints;
    j=1;
```

```
  /* bit reversal section */
   for (i=1; i <= n ; i+=2)
    {
      if (j > i)
       {
/* Note: several statements have been commented out for the case
         where input imaginary values are always zero.  If this is
         not true, then these statements must be used.
*/
          tempr = spc_buf[j];
/*          tempi = spc_buf[j+1];    */
          spc_buf[j] = spc_buf[i];
/*          spc_buf[j+1] = spc_buf[i+1];   */
          spc_buf[i] = tempr;
/*          spc_buf[i+1] = tempi;   */
        }
      m=n/2;
      while ( m >= 2 && j > m )
        {
          j=j-m;
          m=m/2;
        }
      j=j+m;
    }

  /* compute the butterflies */

    mmax=2;
    while ( n > mmax )
      {
      istep= 2 * mmax;
      theta = 2.0 * pi /(float)mmax;
      wsin = sin(0.5 * theta);
      wpr = -2.0*wsin*wsin;
      wpi = sin(theta);
      wr = 1.0;
      wi = 0.0;
      for (m=1; m <= mmax; m+=2)
        {
          for (i=m; i <= n; i=i+istep)
            {
              j=i+mmax;
              tempr = wr*spc_buf[j] - wi*spc_buf[j+1];
              tempi = wr*spc_buf[j+1] + wi*spc_buf[j];
              spc_buf[j] = spc_buf[i] - tempr;
              spc_buf[j+1] = spc_buf[i+1] - tempi;
              spc_buf[i] = spc_buf[i] + tempr;
              spc_buf[i+1] = spc_buf[i+1] + tempi;
            }
          wtemp = wr;
          wr = wr*wpr - wi*wpi + wr;
          wi = wi*wpr + wtemp*wpi + wi;
```

```
         }
         mmax=istep;
      }
  }
/*********************** end of RFFT *********************************/
/*********************** function fburst *****************************/
/* FBURST

    This routine will find the center burst of an interferogram array.
    The routine is a function call as the burst value is returned.

    routines called:
       NONE
--------------------------------------------------------------------*/
int fburst(raw_buf,ipoints)
/* The following global parameters are:
      raw_buf  - the interferogram array
      ipoints  - the number of points in interferogram
*/
float raw_buf[];
int ipoints;
 {
   int i,max_loc, min_loc;
   float max_val=0.0, min_val=0.0;

/*   printf ("ipoints in fburst= %04d\n",ipoints);
   printf ("raw_buf[50]= %10.5f\n",raw_buf[50]);*/
   for (i=1;i <= ipoints; i++)
    if (raw_buf[i] > max_val)
      {
        max_val=raw_buf[i];
        max_loc = i;
/*        printf("max_loc= %04d\n",max_loc);  */
      }
    else if (raw_buf[i] < min_val)
      {
        min_val = raw_buf[i];
        min_loc = i;
/*        printf ("min_loc= %04d\n",min_loc);  */
      }

     if (fabs((double) min_val) > max_val)
        return (min_loc);
   else
      return (max_loc);
  }
/*********************** end of FBURST *******************************/
/*********************** function normal *****************************/
/* NORMAL

   This routine is used to normalize the spectral buffer.
```

```
   routines called:
    NONE

--------------------------------------------------------------------*/
void normal (buffer, ipoints)
float buffer[];
{
  int index;
  float ssq = 0.0;

  for (index = 0; index < ipoints; index++)
    ssq += buffer[index] * buffer[index];

  if (ssq > 0.0)
    ssq = ipoints / sqrt (ssq);
  else
    ssq = 1.0;

  for (index = 0; index < ipoints; index++)
    buffer[index] *= ssq;
}
/*********************** end of normal ******************************/
/*********************** function rotate *****************************/
/* ROTATE

   This routine will rotate an interferogram buffer.  The buffer will
   be rotated so that the center burst is in array position 1.

   routines called:
      NONE

-------------------------------------------------------------------*/
void rotate (burst, raw_buf, spc_buf, ipoints)
/* The following parameters are:
    raw_buf  -  the input interferogram buffer
    spc_buf  -  the rotated interferogram buffer
    burst    -  the interferogram center burst array position
    ipoints  -  number of interferogram points is arrays
*/
float raw_buf[], spc_buf[];
int ipoints, burst;
 {
  int oindex, nindex;

  for (oindex=burst, nindex=1; oindex <= ipoints; oindex++, nindex++)
    spc_buf[nindex] = raw_buf[oindex];
/*  nindex-=1;      */
  for (oindex=1; oindex < burst; oindex++)
    {
      spc_buf[nindex] = raw_buf[oindex];
      nindex++;
     }
```

```
  }
/*********************** end of ROTATE ******************************/
/*********************** function draw_axis *************************/
/* DRAW_AXIS

   This routine will draw the axis for either an interferogram or
   spectrum display.

   routines called:
     Microsoft C graphics display routines

-------------------------------------------------------------------*/
void draw_axis (scan,imode)
/* The following parameters are:
     scan  - the scan number
     imode - display mode type; 0=interferogram, 1=spectrum
*/
int scan, imode;
{
  int  i, ih;
  char buffer[80];

  if (imode == 1)
    ih = 150;
  else
    ih = 0;

  _moveto (0, ih+0);    /* Print the X axis */
  _lineto (512, ih+0);
  _moveto (0,150);    /* Print the Y axis */
  _lineto (0,-150);

  for(i = 0; i <= 512; i += 64)  /*Print the X axis tick marks */
    {
    _moveto(i, ih+5);
    _lineto(i, ih+0);
    }

  for(i = 0; i <= 512; i += 32)
    {
    _moveto(i, ih+3);
    _lineto(i, ih+0);
    }

  for(i = 0; i <= 512; i += 16)
    {
    _moveto(i, ih+2);
    _lineto(i, ih+0);
    }

/*  for(i = 150; i > -150; i -= 25)   Print the Y axis tick marks
    {
```

```
    _moveto(-4, i+1);
    _lineto(0, i+1);
    }  */

  /* Label the axis */
  _settextposition(25,36);          /*  X AXIS */
  _outtext (" SCAN # ");
   sprintf(buffer,"%05d",scan);
  _settextposition(25,45);
  _outtext (buffer);
  if (imode == 3)
    {
      _settextposition (25, 8);
      _outtext ("700");
      _settextposition (25, 70);
      _outtext ("1400");
    }

  _settextposition(9,5);            /* Y AXIS */
  _outtext ("A");
  _settextposition(10,5);
  _outtext ("/");
  _settextposition(11,5);
  _outtext ("D");
  _settextposition(13,5);
  _outtext ("u");
  _settextposition(14,5);
  _outtext ("n");
  _settextposition(15,5);
  _outtext ("i");
  _settextposition(16,5);
  _outtext ("t");
  _settextposition(17,5);
  _outtext ("s");

}
/**************************** end of DRAW_AXIS ******************************/
/**************************** function logoega *****************************/
/*  logoega is a function used to create the CBDA logo for EGA graphics.
    The funtion requires two parameters, the x and y coordinates for the
    first letter "C". If the logo coordinates are outside the exceptable
    range, no logo will be plotted.

  author: John Ditillo
  modified by: Bob Kroutil

          logoega is based on the "old" CRDEC logo routine
          written by John T. Ditillo

  date: October 1992  */

void logoega(y,x)
```

```
int y, x;

{
  int xp, yp;

  if (y<23 & y>1 & x<76 & x>2)
    {

    /* draw the logo */
    _settextposition(y,x);
    _outtext ("C");

    _settextposition(y+1,x-1);
    _outtext ("B D");

    _settextposition(y+2,x);
    _outtext ("A");

    /* Calculate first pixel location */
    yp = y * 14 - 16;
    xp = x * 8 - 5;

    /* first benzene */
    _moveto(xp,yp);
    _lineto(xp-8,yp+3);
    _lineto(xp-8,yp+13);
    _lineto(xp,yp+17);
    _lineto(xp+8,yp+13);
    _lineto(xp+8,yp+3);
    _lineto(xp,yp);

    /* second benzene */
    _moveto(xp-8,yp+13);
    _lineto(xp-16,yp+17);
    _lineto(xp-16,yp+27);
    _lineto(xp-8,yp+31);
    _lineto(xp,yp+27);
    _lineto(xp,yp+17);

    /* third benzene */
    _moveto(xp+8,yp+13);
    _lineto(xp+16,yp+17);
    _lineto(xp+16,yp+27);
    _lineto(xp+8,yp+31);
    _lineto(xp,yp+27);

    /* fourth benzene */
    _moveto(xp-8,yp+31);
    _lineto(xp-8,yp+42);
    _lineto(xp,yp+45);
    _lineto(xp+8,yp+42);
    _lineto(xp+8,yp+31);
```

```
  }

}
/************************  end of LOGOEGA ****************************/
/************************  function plotr ****************************/
/* PLOTR

   This routine is used to scale and display the interferogram or
   spectrum.

   routines called:
    Microsoft C graphics routines

-------------------------------------------------------------------*/
void plotr (buf, istps, iendp, imode)
/* The following parameters are:
      buf    - the array buffer to plot
      istps  - the starting point to display
      iendp  - the ending point to display
      imode  - the display mode (0=interferogram, 1=spectrum)
*/
float buf[];
int istps, iendp, imode;
{
  int index, x, y, ih, ip;
  float max, xscale, yscale;

  /* number of points to plot */
  ip = iendp - istps;

  /* find the largest value */
  for (index=istps, max=0.0; index < iendp; index++)
    {
    if ((fabs((double)buf[index])) > max)
      max = (float) (fabs((double)buf[index]));
    }

  /* Calculate the scaling factor */
  xscale = 512.0/ip;
  if (imode == 1)
    {
    yscale = 300.0/max;
    ih = 150;
    }
  else
    {
    yscale = 150.0/max;
    ih = 0;
    }

  /* plot the data */
  _moveto (0, (int) -(buf[istps] * yscale - ih));
```

```
  for (index=1; index < ip; index++)
    {
     x = (int) index * xscale;
     y = (int) -(buf[index+istps] * yscale - ih);
     _lineto (x,y);
    }

}
/**************************** end of PLOTR ****************************/
/********************* function getspc ********************************/
/* GETSPC

  This routine will get up to 4 black body spectra on the disk and
  read them into an array.  The stored spectra are SpectraCalc
  floating point binary format (FSP format - use the input and
  output commands in SpectraCalc).

  routines called:
   NONE

-------------------------------------------------------------------*/
void getspc (spc_bak, ipts, ch)
/* The following parameters are:
  spc_bak - the array that contains the stored disk spectral responses
  ipts    - the number of points in the array
  ch      - a flag to tell which spectrum file to read
*/

float spc_bak[];
int ipts;
char ch;
  {
   int fp3;
   float numpts, firstx, lastx, xunits, yunits, res;
   char afile[20];

/* load the black body spectra */

   if (ch == FSEL5)
     strcpy (afile,"f5.fsp");
   if (ch == FSEL6)
     strcpy (afile,"f6.fsp");
   if (ch == FSEL7)
     strcpy (afile,"f7.fsp");
   if (ch == FSEL8)
     strcpy (afile,"f8.fsp");

   if ((fp3 = open (afile,O_RDONLY|O_BINARY)) >= 0)
     {
      read (fp3, (char *) &numpts, 4);
      read (fp3, (char *) &firstx, 4);
      read (fp3, (char *) &lastx, 4);
```

```
      read (fp3, (char *) &xunits, 4);
      read (fp3, (char *) &yunits, 4);
      read (fp3, (char *) &res, 4);

      if ( read (fp3, spc_bak, 4 * ipts) != 4 * ipts)
        printf("\nUnable to read disk stored black body file.\n");
      close (fp3);
      }
    else
     {
      _settextposition (1,20);
      _outtext ("===> ERROR - disk file .fsp does not exist !!!! <===");
     }
  }
/********************** end of getspc *********************************/
/********************** function errcod ******************************/
/* ERRCOD

   This routine will find out if the data has an error.

   routines called:
     NONE

-----------------------------------------------------------------*/
int errcod (raw_buf, ipoints, burst, lastpeak)
/* The following parameters are:
     raw_buf - the real valued buffer array to test
     ipoints  - number of points in array
     burst    - the array location of the center burst
     lastpeak - the last array location holding the previous center burst
*/

int burst, lastpeak, ipoints;
float raw_buf[];
   {
     int ercod;

     ercod = 0;

     if (ipoints < 1024)
       ercod = 1;
     if (fabs(raw_buf[burst]) >= 32767.)
       ercod = 2;
     if (lastpeak != burst)
       ercod = 3;
     if (burst > 500)
       ercod = 4;
     if (fabs(raw_buf[burst]) <= 8192.)
       ercod = 5;
/*     printf ("raw_data[%04d] = %05d",burst, raw_data[burst]);
       printf ("burst position = %04d", burst);  */
```

```
/* NOTE: error code for bit toggle not yet implemented */

      return (ercod);
    }
/*---------------------------------------------------------------- */
/*      in:     Allow port input during debug.                     */
/*              This is necessary for CV 4.00--the "I" command (port */
/*              input is broken. The circumvention is to include a */
/*              a global function such as in() below, trace at least */
/*              as far as the main() function, then "?in(port)" or */
/*              "?in(port),x" to read port contents.               */
/* --------------------------------------------------------------- */

int in( unsigned port )
{
    int i;
    i = inp(port);
    return i;

} /* in */


/* --------------------------------------------------------------- */
/*      IoDelay:        I/O delay for IBM/AT and clones.           */
/*                                                                 */
/*      This dummy function is used to generate a few clocks of delay */
/*      between consecutive accesses to certain I/O ports. Basically */
/*      the call/return sequence is more than enough. Assembler    */
/*      programs typically use a "JMP SHORT $+2" instruction, but  */
/*      the MSC7 inline assembler doesn't seem to handle the "$"   */
/*      token very well. The delay is necessary on IBM AT machines */
/*      and true compatibles.                                      */
/*                                                                 */
/*      Needless to say, allowing this function to be inlined would */
/*      be a bad idea...                                           */
/* --------------------------------------------------------------- */

static void near IoDelay(void)
{
   ;
} /* IoDelay */

/* --------------------------------------------------------------- */
/*      GetDmaBuffer:   Allocate a byte-DMA compatible buffer      */
/*                                                                 */
/*      A byte DMA buffer cannot cross a 64K-byte absolute address */
/*      boundary.                                                  */
/*                                                                 */
/*      Returns pointer to buffer if successful, NULL otherwise.   */
/* --------------------------------------------------------------- */

void far *GetDmaBuffer(long Size)
```

```
{
    #define MaxTries 16          /* Maximum attempts before failure      */

    void         far *failed[MaxTries],
                  far *try,
                  far *retry;
    unsigned     begoff, endoff;
    int          i, nfail=0;

    if (Size>MAXDMA || Size<=0) return NULL;

    for (;;)                                /* Repeat until explicit break: */
    {
            try = malloc((size_t)Size);
            if ( try==NULL ) break;

/* Test for 64K block wraparound:                                       */

            begoff = (FP_SEG(try) << 4) + FP_OFF(try);
            endoff = begoff + (unsigned)Size - 1;
            if (endoff >= begoff) break;    /* Success if all in 1 block    */

/* Current attempt crosses boundary, retry if failed list not full:     */

            if (nfail == MaxTries)
            {
                free(try);
                try = NULL;
                break;
            }

/* Resize current try to end on 64K absolute boundary and add it to     */
/* the failed list:                                                     */

            retry = realloc(try, 1+~begoff);
            if ( retry != NULL )
                try = retry;
            failed[nfail++] = try;
    }

/* Arrive here via explicit break. Free failed attempt pointers, if     */
/* any and exit. The try variable has been set to a pointer on success  */
/* or to NULL on error.                                                 */

    for( i=0; i<nfail; ++i )
    {
            free( failed[i] );
    }

    return try;

#undef MaxTries                             /* Undefine "local" macros      */
```

```
} /* GetDmaBuffer */


/* ---------------------------------------------------------------- */
/*      StartDma:       Start a DMA operation.                      */
/*                                                                  */
/*      This is a cut-down version to do input only, specifically   */
/*      using DMA info in MidGbl structure.                         */
/* ---------------------------------------------------------------- */

void StartDma(void)
{
    long        addr = PtrToLong(MidGbl.DmaBuffer);
    int         size = (int)MidGbl.DmaSize;
    unsigned    ch   = 2*MidGbl.DmaChannel;

    DisableDma(MidGbl.DmaChannel);
    IoDelay();                          /* Wait a few CPU clocks      */
    outp(DMA_MODE, 0x44+MidGbl.DmaChannel);
                /* DMA Mode: single-block,      */
                /*   increment address,         */
                /*   no autoinitialize,         */
                /*   "write transfer" -> cpu    */
    IoDelay();                          /* Wait a few CPU clocks      */

    outp(DMA_CLRF,0);                   /* Set to receive LSB first   */
    IoDelay();                          /* Wait a few CPU clocks      */

    outp(DMA_CTR+ch, (int)size);        /* Send byte count            */
    IoDelay();                          /* Wait a few CPU clocks      */
    outp(DMA_CTR+ch, (int)size >> 8);
    IoDelay();                          /* Wait a few CPU clocks      */

    outp(DMA_ADDR+ch, (int)addr);       /* Send address               */
    IoDelay();                          /* Wait a few CPU clocks      */
    outp(DMA_ADDR+ch, (int)addr >> 8);
    IoDelay();                          /* Wait a few CPU clocks      */

    outp(MidGbl.DmaPageReg, (int)(addr>>16));
                /* Set page reg to top 8 bits   */
    IoDelay();                          /* Wait a few CPU clocks      */

    EnableDma(MidGbl.DmaChannel);       /* Finally, enable DMA        */

} /* StartDma */


/* ---------------------------------------------------------------- */
/*      SetIrqEnable:   Set/Reset IRQ enable status for specified   */
/*                      channel.                                    */
/*                                                                  */
/*      Please note that the sense of the "Enable" argument is a C- */
```

```
/*      style boolean. Nonzero, or "true", enables the channel. This     */
/*      is opposite from the 8259 mask register, where a 1 disables      */
/*      the channel and 0 enables.                                        */
/* -------------------------------------------------------------------- */

void SetIrqEnable(
    int         IrqNumber,      /* Interrupt channel, 0-15               */
    int         Enable)         /* New enable status for this channel    */
                 /*   0 = disable interrupts              */
                 /*   nonzero = enable interrupts         */
{
    unsigned    port;
    int         mask, val;

    if (IrqNumber < 8)
    {
            port = PIC1_MASK;           /* Primary 8259 port             */
            mask = 1 << IrqNumber;
    }
    else
    {
            port = PIC2_MASK;           /* Secondary 8259 port           */
            mask = 1 << (IrqNumber-8);
    }

    val = inp(port) | mask;             /* Set to mask disable           */
    if (Enable) val -= mask;            /* Set to enable if requested    */
    outp(port, val);                    /* Update port                   */

} /* SetIrqEnable */


/* -------------------------------------------------------------------- */
/*      MidAqStartScan: Start new data collect operation                 */
/*                                                                       */
/*      This is a skeleton of what is needed to begin a new data         */
/*      scan, or series of accumulated scans, on the Midac FT-IR.        */
/* -------------------------------------------------------------------- */

void MidAqStartScan(void)
{
    SetIrqEnable(MidGbl.IrqNum, 0);     /* Disable interrupt channel     */
    IoDelay();                          /* Wait a few CPU clocks         */
    DisableDma(MidGbl.DmaChannel);      /* Disable DMA channel           */
    IoDelay();                          /* Wait a few CPU clocks         */

    StartDma();                         /* Start DMA channel             */

    SetIrqEnable(MidGbl.IrqNum, 1);     /* Enable interrupt channel      */

/* Set gain and retrace interferometer:                                  */
```

```
    CmdOut( MidGbl.GainPort | MIDC_EOS | MIDC_IRQ );
                /* Start IRQ clear pulse*/
    IoDelay();                                    /* Wait a few CPU clocks*/
    CmdOut( CmdIn() &~(MIDC_EOS + MIDC_IRQ) );   /* End IRQ clear pulse, */
                /* Start retrace pulse  */
    IoDelay();                                    /* Wait a few CPU clocks*/
    while (inp(MID_STAT) & MIDS_FLYBK);           /* Wait for turnaround  */
    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ));    /* End retrace pulse    */
    IoDelay();                                    /* Wait a few CPU clocks*/

    /* Note: May need to insert delay here, 10-20ms, to allow for        */
    /* hardware bug in Midac interface causing early DMA requests.       */
                _asm xor  cx,cx
        here:   _asm loop here

    MidGbl.DmaActive = 1;              /* Set global DMA status flags  */
    MidGbl.DmaDone = 0;

    CmdOut( CmdIn() | MIDC_DMA );      /* Enable DMA at interface      */

} /* MidAqStartScan */


/* ------------------------------------------------------------------ */
/*      MidAqDmaDone:   Interrupt Handler for DMA completion           */
/*                                                                     */
/*      This version simply notes DMA completion, retraces the         */
/*      interferometer, and disables DMA at both the 8237 and at       */
/*      the Midac interface board.  This would be the natural place    */
/*      to insert co-add logic for averaging interferograms.           */
/* ------------------------------------------------------------------ */

void _cdecl _interrupt far MidAqDmaDone(void)
{
    MidGbl.DmaDone = 1;                /* Note DMA completion          */

    CmdOut( CmdIn() &~MIDC_DMA );      /* Disable DMA at interface     */
    DisableDma(MidGbl.DmaChannel);     /*  then disable channel        */
    IoDelay();                         /* Wait a few CPU clocks        */

/* Retrace interferometer:                                             */

    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ) );   /* Start IRQ clear pulse*/
    CmdOut( CmdIn() &~(MIDC_EOS + MIDC_IRQ) );   /* End IRQ clear pulse, */
                /* Start retrace pulse  */
    _enable();                                    /* Interrupts on now    */
    while (inp(MID_STAT) & MIDS_FLYBK);           /* Wait for turnaround  */
    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ));    /* End retrace pulse    */

    /* This is the place to put co-add logic and possibly start the      */
    /* DMA controller for a new scan. Note that the instrument will      */
    /* scan anyway--the decision is whether or not to collect the data. */
```

```
    /* Note: May need to insert delay, 10-20ms, to allow for          */
    /* hardware bug in Midac interface, if another scan is to be      */
    /* started here.                                                  */

    outp(PIC1_CMD, PICC_EOI);           /* Issue EOI to master          */
    IoDelay();                          /* Wait a few CPU clocks        */
    if (MidGbl.IrqNum > 7)              /* If interrupt is on slave     */
            outp(PIC2_CMD, PICC_EOI);       /*   then issue secondary EOI   */

} /* MidAqDmaDone */


/* ------------------------------------------------------------------ */
/*      MidAqSetGain:   Set Signal Gain                                */
/*                                                                     */
/* ------------------------------------------------------------------ */

int MidAqSetGain(int SignalGain)
{
    int gainport = ((~SignalGain << MIDC_GSHIFT) & MIDC_GMASK);
    int oldgain = MidGbl.GainVal;

    if (SignalGain<0 || SignalGain>7)
            return -1;

    CmdOut(gainport | (CmdIn() & ~MIDC_GMASK));
    MidGbl.GainVal = SignalGain;
    MidGbl.GainPort = gainport;
    return oldgain;

} /* MidAqSetGain */


/* ------------------------------------------------------------------ */
/*                                                                     */
/*      MidAqTerm:      Data collect termination                       */
/*                                                                     */
/*      This function is not explicitly called, but is called at       */
/*      program termination via the atexit() facility. The primary     */
/*      task is to disable DMA and the terminal count interrupt and    */
/*      restore the IRQ vector.                                        */
/* ------------------------------------------------------------------ */

void MidAqTerm(void)
{

    SetIrqEnable(MidGbl.IrqNum, 0);     /* Disable interrupt channel    */
    DisableDma(MidGbl.DmaChannel);      /* Disable DMA channel          */
    CmdOut(MIDC_EOS);                   /* Reset the interferometer     */
    IoDelay();                          /* Wait a few CPU clocks        */

    if (MidGbl.OldIrqVec != NULL)
```

```
    {
            _dos_setvect(MidGbl.IrqVecNo, MidGbl.OldIrqVec);
            MidGbl.OldIrqVec = NULL;
    }

} /* MidAqTerm */


/* ------------------------------------------------------------------ */
/*      MidAqInit:      Initialize Midac interface for data collect     */
/*                                                                      */
/*      The arguments to this function provide for setup parameters    */
/*      and/or nonstandard interface board configurations. Each is     */
/*      either a nonnegative integer value, or -1 to use the           */
/*      predefined default value.                                      */
/*                                                                      */
/*      The first two arguments (DmaChannel, IrqNumber) describe the   */
/*      configuration of the Midac interface board. Current interface  */
/*      boards are hardwired for DMA channel 1 and are jumper          */
/*      selectable to use either IRQ2 or IRQ3. Other options could     */
/*      conceivably be possible for unusual custom requirements.       */
/*      In general, however, such a modified interface board would     */
/*      be incompatible with existing SpectraCalc and LabCalc drivers. */
/*                                                                      */
/*      The buffer size argument (MaxPoints) is necessary to allocate  */
/*      a DMA buffer. This buffer has the hardware-enforced            */
/*      requirement to not cross a 64K-byte absolute memory boundary.  */
/*      This is the strictest dynamic allocation requirement in a      */
/*      typical data collect application, and should be done first.    */
/*      If co-addition of interferograms is to be performed, this is   */
/*      might be a good place to allocate an accumulator buffer as     */
/*      well.                                                          */
/*                                                                      */
/*      The gain argument (SignalGain) provides the initial signal     */
/*      gain level for programming the interface. This value is        */
/*      subject to change during program operation, but some initial   */
/*      value is required.                                             */
/* ------------------------------------------------------------------ */

int MidAqInit(
    int         DmaChannel,     /* DMA channel number, 0-3              */
    int         IrqNumber,      /* PC/ISA interrupt channel number      */
    int         SignalGain,     /* Signal gain level, 0-7               */
    int         MaxPoints)      /* Max data points in collect buffer    */
{
    int         i, dmachan, irqnum, maxpts, gainval, gainport;

/* Translate and validate input paramters...                           */

    dmachan     = DmaChannel>=0 ? DmaChannel : DMA;
    irqnum      = IrqNumber >=0 ? IrqNumber  : IRQ;
    gainval     = SignalGain>=0 ? SignalGain : GAIN;
```

```
    maxpts       = MaxPoints>=0  ? MaxPoints  : BUFPTS;

    if (dmachan != DMA) return -1;       /* ***temp*** need to know page */
                 /* register addresses for other */
                 /* DMA channels to generalize   */
                 /* this for other byte channels */

    if (dmachan<0 || dmachan>3)
            return -1;
    if (irqnum<0 || irqnum>15)
            return -1;
    if (gainval<0 || gainval>7)
            return -1;
    if (maxpts<1 || maxpts>(MAXDMA / 2))
            return -1;

/* Bring the hardware interface to idle state:                           */

    gainport = (~gainval << MIDC_GSHIFT) & MIDC_GMASK;
                 /* Compute inverted gain val    */
    MidGbl.GainVal      = gainval;       /* Save requested gain          */
    MidGbl.GainPort     = gainport;      /* Save port image              */

    CmdOut(gainport | MIDC_EOS);         /* Set gain, DMA off, and       */
                 /*   EOS,IRQ strobes off.      */

    SetIrqEnable(irqnum, 0);             /* Disable interrupt channel    */
    DisableDma(dmachan);                 /* Disable DMA channel          */
    IoDelay();                           /* Wait a few CPU clocks        */

/* Initialize DMA:                                                       */

    MidGbl.DmaDone      = 0;
    MidGbl.DmaActive    = 0;
    MidGbl.MaxPoints    = maxpts;
    MidGbl.DmaChannel   = dmachan;
    MidGbl.DmaPageReg   = DmaPageTable[dmachan];
    MidGbl.DmaSize      = (long)maxpts * sizeof(unsigned short);
    MidGbl.DmaBuffer    = GetDmaBuffer(MidGbl.DmaSize);
    if (MidGbl.DmaBuffer == NULL)
            return -1;

    for (i=0; i<maxpts; ++i)             /* Put recognizable null data   */
            MidGbl.DmaBuffer[i] = 0xEEEE;    /*   in buffer for debug        */

/* Initialize IRQ channel                                                */

    MidGbl.IrqNum       = irqnum;
    MidGbl.IrqVecNo     = (irqnum<8 ? 0x08 : 0x68) + irqnum;
    MidGbl.OldIrqVec    = _dos_getvect(MidGbl.IrqVecNo);
    _dos_setvect(MidGbl.IrqVecNo, MidAqDmaDone);
```

```
    atexit(MidAqTerm);

    return 0;

} /* MidAqInit */
```

APPENDIX E

DATA CONVERSION PROGRAM (SCCONV)

```
/**************************************************************************/
/* program SCCONV                                          version 2.0

   Date:  1 June 1993

   Author:  Bob Kroutil

   This program will read a series of SpectraCalc files and
store the results into a binary format interferogram file.

Note: A directory listing file is needed by this program for the
      filename input.  This file can be created by the DOS command
      "dir *.spc >dirfile" ... where dirfile is the name of the
      directory file to be used as input to this program.  This
      file should be edited to remove the starting and ending
      information.  A listing example of this directory file
      can be shown in the supplied file "listfile".


*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <graph.h>
#include <string.h>
#include <math.h>
#include <errno.h>
#include "headers.def"
#include "scalc.def"
#include "exscconv.def"

#define GH_LENGTH   512
#define SH_LENGTH    64
#define SCALC       256
#define PMODE      0644
#define MAXLINE      80
#define EFILE        46

main ()
  {
   char dirname[58], lname1[58], lname2[4], lname3[58], lnamef[58];
   char comm1[64], comm2[64], comm3[64], comm4[64], sensid[20], opernam[10];
   char idate[10], itime[10], lnameh[10], lnamea[10], ieof, itr[2];
   char lname5[2], lname6[2];
   int fp2, sptr, iscan, itemp, j, jj, igain;
   long int itempl, intsize;
   size_t icount2;
   double rmaxv4, rmaxv3, rmaxv2, rmaxv1;
   FILE *fd, *fopen();
   extern long intl_buffer[];
   extern int int_buffer[];
```

```
   struct global_header gh;
   struct scan_header sh;
   struct spec_header ah;

/* initialize the global header and scan header */
  memset (&gh, 32, GH_LENGTH);
  memset (&sh, 32, SH_LENGTH);
  memset (comm1, 32, 64);
  memset (comm2, 32, 64);
  memset (comm3, 32, 64);
  memset (comm4, 32, 64);
  memset (sensid, 32, 20);
  memset (lnamea, 32, 10);
  memset (itr, 0, 2);
  strcpy (lname5, "/");
  strcpy (lname6, ":");

/* user input section */
   _clearscreen (_GCLEARSCREEN);
   printf("\nSCCONV   -     Interferogram Data Conversion Program          V
2.0");
   printf("\n\n Input the directory listing filename: ");
   scanf ("%s",dirname);
   printf("\n Input the interferogram output filename: ");
   scanf ("%s",lnamef);
   printf("\n Input the header name for the file: ");
   scanf ("%s",lnamea);

/* get the interferometer scan parameters */
   printf("\n Input the scan speed of the interferometer: ");
   scanf("%lf",&gh.scan_speed);
   printf("\n Input the mirror velocity of the interferometer: ");
   scanf("%lf",&gh.mirror_velocity);
   printf("\n Input the sampling frequency of the interferometer: ");
   scanf("%lf",&gh.sample_freq);
   printf("\n Input the starting transform frequency: ");
   scanf("%lf",&gh.start_freq);
   printf("\n Input the ending transform frequency: ");
   scanf("%lf",&gh.stop_freq);
   printf("\n Input the number of zero crossings per sampled point: ");
   scanf("%d",&gh.zercross);
   printf("\n Input the data collection mode: ");
   scanf("%d",&gh.collect_mode);

/* input four lines for the global comments */
   printf("\n Input four lines for comments:\n");
   gets(comm1);
   printf(">>");
   gets(comm1);
   printf(">>");
   gets(comm2);
   printf(">>");
```

```
   gets(comm3);
   printf(">>");
   gets(comm4);
   printf("\n Enter the sensor ID name for interferogram header: ");
   gets(sensid);

/* create the interferogram output file and open for writing */
   if (( fp2 = creat(lnamef, PMODE)) < 0 )
     {
      printf("\n\SCCONV\" is unable to create %s\n", lnamef);
      exit(2);
     }
   if (( fp2 = open (lnamef, O_WRONLY|O_BINARY)) < 0 )
     {
      printf("\n\SCCONV\" is unable to open %s\n", lnamef);
      exit (3);
     }

/* open the directory listing file and loop to read the SpectraCalc
   file names */
   if (( fd = fopen(dirname, "r")) == NULL )
    {
     printf("\nERROR - can not open directory listing file.");
    }

/* loop to read each interferogram filename in the directory file */
   iscan = 0;
tloop:
   iscan++;

/* read one directory file record */
   memset(lname3, 0, 58);
   memset(lnamel, 0, 58);
   memset(lnameh, 0, 10);
   fgets (lnameh, MAXLINE, fd);
   icount2 = 9;
   memcpy (&lnamel, &lnameh, icount2);
   strcpy (lname2, ".SPC");
   strcat (lnamel, lname2);
   _fullpath(lname3, lnamel, 58);
   icount2 = 1;
   memcpy (&ieof, &lnamel, icount2);

/* if at the end of the file then stop and close all files */
   if (ieof == EFILE)
     {
      printf("\n\nTotal Number of Interferograms Converted ===> %4d\n",
              iscan-1);
      gh.stop_scan = iscan - 1; /* insert the number of scans in header */

/* insert the correct stop time into the global header */
/* insert the correct hour into the global header */
```

```
        icount2 = 1;
        itemp = 0;
        memcpy( &itemp, &ah.ihour, icount2);
        sprintf( gh.stop_time, "%2d", itemp);
        memcpy( gh.stop_time+2, lname6, icount2);

/* insert the correct minute into the stop time */
        memcpy( &itemp, &ah.iminute, icount2);
        sprintf( gh.stop_time+3, "%2d", itemp);

/* dummy the seconds into the subfile header */
        memcpy( gh.stop_time+5, lname6, icount2);
        itemp = 0;
        sprintf( gh.stop_time+6, "%1d", itemp);
        sprintf( gh.stop_time+7, "%1d", itemp);

/* write the global header and close all files before exit */
        lseek (fp2, 0L, 0); /* rewind the file to the beginning */
        write (fp2, &gh, GH_LENGTH);  /* write global header */
        close (fp2);
        fclose (fd);
        exit (1);
       }

/* tell the user what file we are converting */
  printf("\n------------------------------------------------------------------");
  printf("\nwriting interferogram # %04d ==> reading filename :
%s",iscan,lname3);


/* open the SpectraCalc format data files */
   if ((sptr = open (lname3, O_RDONLY|O_BINARY)) < 0)
     {
      printf("\n Unable to open the SpectraCalc file %s\n", lname3);
      exit (4);
     }

/* read the SpectraCalc 256 byte header record */
   if ( read ( sptr, &ah, SCALC) != SCALC)
     {
      printf("\nERROR - can not read SpectraCalc header for file
%s\n",lnamef);
      exit (5);
     }

/* check to see if too many points are in the file */
   itemp = (int) ah.npts;
   if (itemp > MAXPOINTS)
     {
      printf("\nERROR: > %d points in file ... # points= %d\n",MAXPOINTS,
             itemp);
      printf("\nfilename= %s\n",lname3);
```

```
      exit(9);
     }

/*-----------------------------------------------------------------*/
/* convert the SpectraCalc global header information and subfile information
*/
/* convert the global information */
   if (iscan == 1)
     {
/* find the interferogram size to write to disk */
      intsize = (int) ah.npts;
/* set the filename into the global header */
      icount2 = 10;
      memcpy (&gh.filename, &lnamea, icount2);
/* insert other global header information */
      gh.integer_type = 1;
      gh.scan_size = (int) ah.npts;
      memcpy(&itr, &ah.iresol, 1);
      gh.resolution = (double) (atoi (itr));
      gh.max_wav = gh.stop_freq * (float)gh.zercross;
      gh.zercross = gh.zercross * 100;

/* input the weather information into the header if needed */
      gh.ambient_temp = 0;
      gh.bar_pressure = 0.0;
      gh.humidity = 0;
      gh.wind_speed = 0;
      gh.wind_direction = 0;
      gh.sensor_direction = 0;
      gh.precip_code = 0;

/* insert the correct month into the global header */
      icount2 = 1;
      itemp = 0;
      memcpy( &itemp, &ah.imonth, icount2);
      sprintf( gh.date, "%2d", itemp);

/* insert the day into the global header  */
      memcpy( gh.date+2, lname5, icount2);
      memcpy( &itemp, &ah.iday, icount2);
      sprintf( gh.date+3, "%2d", itemp);

/* insert the year into the global header */
      memcpy( gh.date+5, lname5, icount2);
      itemp = ah.iyear - 1900;
      sprintf( gh.date+6, "%2d", itemp);

/* insert the starting time into the global header */
      memcpy( &itemp, &ah.ihour, icount2);
      sprintf( gh.start_time, "%2d", itemp);
      memcpy( gh.start_time+2, lname6, icount2);
```

```
/* insert the correct minute into the header */
      memcpy( &itemp, &ah.iminute, icount2);
      sprintf( gh.start_time+3, "%2d", itemp);

/* dummy the seconds information into the subfile header */
      memcpy( gh.start_time+5, lname6, icount2);
      itemp = 0;
      sprintf( gh.start_time+6, "%1d", itemp);
      sprintf( gh.start_time+7, "%1d", itemp);

/* input the sensor ID name into the global header */
      icount2 = 20;
      memcpy (&gh.sensor_id, &sensid, icount2);

/* input the operators name into the header */
      memset(opernam, 0, 10);
      icount2 = 10;
      strcpy (opernam,"          ");
      memcpy (&gh.operator, &opernam, icount2);

/* insert the comment data into the global header */
      icount2 = 64;
      memcpy (&gh.comm1, comm1, icount2);
      memcpy (&gh.comm2, comm2, icount2);
      memcpy (&gh.comm3, comm3, icount2);
      memcpy (&gh.comm4, comm4, icount2);
              }
/*----------------------------------------------------------------------*/

/* check to see if current interferogram has the same number of points
   as all other interferograms */
  itemp = (int) ah.npts;
  if (itemp != intsize)
    {
     printf("\nERROR: number of points in interferogram file not the");
     printf("\n       same as other interferograms...\n");
     printf("filename= %s\n",lname3);
     exit(9);
    }

/* write the interferogram global header information to disk */
  if (iscan == 1)
    {
     if (write ( fp2, &gh, GH_LENGTH) != GH_LENGTH)
        {
         printf("\nERROR - Unable to write global header of output
interferogram file.");
         exit (6);
        }
    }

/* read the SpectraCalc data from disk */
```

```
  if ( read ( sptr, int_buffer, 4 * intsize) != 4 * intsize)
    {
     printf("\nERROR - Unable to read SpectraCalc data from disk ");
     exit (7);
    }

/* convert the SpectraCalc data to binary format for interferogram file */
/* swap the 16 bits around for the correct word order */
   for ( j = 0; j < 2*intsize; j+=2)
    {
     itemp = int_buffer[j+1];
     int_buffer[j+1] = int_buffer[j];
     int_buffer[j] = itemp;
    }
   icount2 = 4 * intsize;
   memcpy (&intl_buffer, &int_buffer, icount2);

/* calculate/convert the correct gain and scaling for the interferogram */
/*   printf("\nah.igain=%d\n",ah.igain); */
   itempl = 1;
   if (ah.igain <= 16)
     rmaxv1 = (double) (itempl << ( 32 - ah.igain ));
   else
    rmaxv1 = 1.0;
   if (ah.igain <= 7)
     rmaxv1 = (double) (itempl << 16);
   rmaxv3 = 0.0;
   for ( j = 0; j < intsize; j++)
     {
      rmaxv2 = ((double) intl_buffer[j])/rmaxv1;
      rmaxv3 = max ( rmaxv2, rmaxv3);
     }
   if (ah.igain <=16)
     {
     rmaxv2 = (log ((double) (itempl << 8))) / (log (rmaxv3));
     igain = (int) rmaxv2;
     rmaxv4 = (double) (itempl << (igain));
     }
   if (ah.igain > 16)
     {
     igain = ah.igain - 32;
     rmaxv4 = 1.0;
     }
   if (ah.igain <=7 )
     {
     igain = ah.igain - 16;
     rmaxv4 = 1.0;
     }
   for ( j=0; j < intsize; j++)
     {
      rmaxv2 = ((double) intl_buffer[j])/rmaxv1;
      rmaxv3 = rmaxv2 * rmaxv4;
```

```
/*      printf("\nrmaxv3=%f",rmaxv3); */
      int_buffer[j] = (int) rmaxv3;
     }
/*----------------------------------------------------------------------*/
/* insert the correct interferogram subfile header information */
   sh.scan_number = iscan;    /* input the scan number */
   sh.peak_location = (int) ah.scheadl;  /* input the peak location */
   sh.gain = igain;           /* input the A/D gain */
   sh.coadd = 1;              /* input the # of coadded interferograms */
   icount2 = 9;
   memcpy (&sh.filename, &lnameh, icount2); /* input filename */
   sh.error = 0; /* assume all error codes=0 for the SpectraCalc files */

/* insert the correct time into the subfile header */
   icount2 = 1;
   itemp = 0;
   memcpy( &itemp, &ah.ihour, icount2);
   sprintf( sh.scan_time, "%2d", itemp);
   memcpy( sh.scan_time+2, lname6, icount2);

/* insert the correct minute into the header */
   memcpy( &itemp, &ah.iminute, icount2);
   sprintf( sh.scan_time+3, "%2d", itemp);

/* dummy the seconds information into the subfile header */
   memcpy( sh.scan_time+5, lname6, icount2);
   itemp = 0;
   sprintf( sh.scan_time+6, "%1d", itemp);
   sprintf( sh.scan_time+7, "%1d", itemp);

/* if no peak position put into the SpectraCalc header, then find it */
   if (sh.peak_location == 0)
     {
      itemp = 0;
      for ( j = 0; j < intsize; j++)
       {
        jj = abs (int_buffer[j]);
        if (itemp < jj)
          {
           itemp = abs (int_buffer[j]);
           sh.peak_location = j+1;
          }
       }
     }

/*-------------------------------------------------------------------*/

/*-------------------------------------------------------------------*/
/* print the spectracalc header information out to the screen */
   printf("\ninterferogram number          = %d",sh.scan_number);
   printf("\ninterferogram gain            = %d",ah.igain);
   printf("\nnumber of points              = %ld",intsize);
```

```
   printf("\nresolution                        = %f",gh.resolution);
   printf("\nmaximum peak location             = %d",sh.peak_location);
   printf("\ncomment 1 >> %s",ah.scomm1);
   printf("\ncomment 2 >> %s",ah.scomm2);
   printf("\ncomment 3 >> %s",ah.scomm3);
/*------------------------------------------------------------------*/

/* write the subfile header information to the interferogram disk file */
   write (fp2, &sh, SH_LENGTH);

/* write the interferogram data to disk */
  if (write ( fp2, int_buffer, 2*intsize) != 2*intsize)
    {
     printf("\nERROR - Unable to write the interferogram data to disk");
     exit (8);
    }
   close (sptr);

   goto tloop;
 }




^Z
```

Blank

APPENDIX F

DATA CONVERSION PROGRAM (CONVINTF)

```
/***************************** program CONVINTF ******************************/
/*
 program CONVINTF                                          version 1.5

  This program will convert an interferogram from a sequential file
 type (as created by program MIDCOL) to multiple files that can
 be read using the SpectraCalc binary floating point format.  The
 program can also output a SpectraCalc data file format file.

  author: Bob Kroutil

  date: May 1993

-----------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <graph.h>
#include <string.h>
#include <math.h>
#include <errno.h>
#include "headers.def" /* include headers for interferogram file */
#include "scalc.def"   /* include the headers for spectracalc files */
#include "exconv.def"  /* external arrays */

#define GH_LENGTH  512
#define SH_LENGTH   64
#define SCALC      256

main ()
 {
  char dirname[58], lname1[30], lname2[30], lname3[4], lnamef[58], lname4[6];
  int fp2, sptr, istart, istop, scan, iscan, j;
  int itype, igainl, jj, itemp;
  float xstart, xstop, xunits, yunits, resol, gain, numpts, tmaxv, smaxv;
  float schead[8], xbeg, numptsl, xend;
  long int position, itempl, intsize;
  char ivn[1], ihex[1], ixtype[1], iytype[1], imonth[2], atime[4];
  char iday[2], ihour[2], iminute[2];
  char iresol[8];
  double tmaxvl, smaxvl, rmaxvl, rmaxv2;
  size_t index=30, icount2;
  extern int int_buffer[];
  extern long intl_buffer[];
  extern float intf_buffer[];
  struct global_header gh;
  struct scan_header sh;
  struct spec_header ah;

/* initialize spectracalc data header constants */
  icount2 = 1;
```

```
  memset(ivn, 0, icount2);
  memset(ihex, 77, icount2);
  memset(ixtype, 0, icount2);
  memset(iytype, 1, icount2);

/* ask the user to input the data file names */
  _clearscreen (_GCLEARSCREEN);
  printf("\nCONVINTF  -          Midac Data Conversion Program
Version 1.5\n");
  printf("\nInput the interferogram file name to read : ");
  scanf("%s",dirname);
  printf("\nInput a partial character name for output interferogram filename :
");
  scanf("%s",lname1);
  printf("\nInput the starting interferogram to convert : ");
  scanf("%d",&istart);
  printf("\nInput the ending interferogram to convert : ");
  scanf("%d",&istop);
  istart = istart - 1;
  istop = istop - 1;
  printf("\nInput the menu format type number for interferogram data
conversion:\n");
  printf(" 1 = floating point   2 = SpectraCalc format \n");
  scanf("%d",&itype);

/* if data type not correct, then stop the program and print error message */
   if (itype <= 0 || itype >= 3)
     {
     printf("\nKEYBOARD INPUT ERROR - data type to output does not exist.\n");
     exit(9);
     }

/* open the binary file and read the global header information */
  if ((fp2 = open (dirname, O_RDONLY|O_BINARY)) < 0)
    {
     printf ("\n\"CONVINTF\" is unable to open %s\n",dirname);
     exit(2);
    }
   if (read (fp2, &gh, GH_LENGTH) != GH_LENGTH)
     {
      printf("\nERROR - Unable to read global header of input interferogram
file.\n");
      exit(3);
      }

/* find the interferogram size and check to see if too many data points
   are in file */
   intsize = gh.scan_size;
   if (intsize > MAXPOINTS)
     {
      printf("\nERROR: > %d data points in file - #points=%ld\n",MAXPOINTS,
             intsize);
```

```
      exit(9);
     }

/* tell the user how many interferograms in the interferogram file */
  printf("\nLast interferogram number in input data file ===> %4d\n",
         gh.stop_scan);

  if (istop > gh.stop_scan || istart > gh.stop_scan)
    {
      printf ("\n ERROR - interferogram to convert > interferograms
present\n");
      exit(4);
     }
  if (istop < 0 || istart < 0)
    {
     printf ("\nERROR - interferogram number out of range.\n");
     exit(4);
    }

/* position the disk file for the first file to read */
   position = ((long) istart) * 2112L + 512L;
   lseek (fp2, position, 0);

/* loop to read interferograms and store each to disk with the appropriate
   file name  */
  for (scan = istart; scan <= istop; scan++)
    {
     iscan = scan+1;

/* read the binary interferogram subfile */
      if (read (fp2, &sh, SH_LENGTH) !=SH_LENGTH)
        {
         printf("\nERROR - Unable to read the subfile header\n");
         exit(5);
        }

/* read the binary interferogram data points */
      if (read(fp2, int_buffer, 2*intsize) != 2*intsize)
        {
         printf("\nERROR - Unable to read the interferogram data\n");
         exit(6);
        }


/* set up the correct path name */
     memset (lnamef, 0, 58);
     memcpy (lname2, lname1, index);
     if (itype == 1)
       strcpy (lname4,".fsp");
     else
       strcpy (lname4,".spc");
     sprintf (lname3, "%04d", iscan);
```

```
     strcat (lname2, lname3);
     strcat (lname2, lname4);
     _fullpath(lnamef, lname2, 58);

/* write out the file number and name to the screen */
    printf("\nreading interferogram # %04d ==> writing filename: %s",
           sh.scan_number,lnamef);

/* try to open the file for writing only */
    if ((sptr = open(lnamef, O_WRONLY|O_BINARY|O_CREAT)) < 0 )
      {
       printf("\nUnable to open the file %s\n",lnamef);
       exit(7);
      }

/* initialize the header data for each binary file */
    numpts = (float) gh.scan_size;
    xstart = 0.0;
    xstop = numpts;
    xunits = 0.0;
    yunits = 1.0;  /* SpectraCalc value for interferogram points */
    resol = gh.resolution;

/************************************************************************/
/* write to the disk if itype = 1 (floating point format) */
   if (itype == 1)
   {

/* gain range the interferogram and store it into the real array */
    if (sh.gain >=0 )
      gain = (float) (1 << sh.gain);
    else
      gain = (float) (1 << (-sh.gain));
    for (j = 0; j < intsize; j++)
      {
      if (sh.gain >=0)
        intf_buffer[j] = ((float)int_buffer[j]) * gain;
      else
        intf_buffer[j] = ((float)int_buffer[j]) / gain;
      }

/* write the interferogram information to disk */
    write (sptr, (char *) &numpts, 4);
    write (sptr, (char *) &xstart, 4);
    write (sptr, (char *) &xstop, 4);
    write (sptr, (char *) &xunits, 4);
    write (sptr, (char *) &yunits, 4);
    write (sptr, (char *) &resol, 4);

    if (write (sptr, intf_buffer, 4*intsize) != 4*intsize)
      {
       printf("\nERROR - Unable to write the interferogram data to disk\n");
```

```
        exit(8);
       }
     }

/**************************************************************************/
/* write to the disk if itype = 2 (SpectraCalc format) */
     if (itype == 2)
     {

/* scale the data for the spectracalc format */
      tmaxv = 0.0;
      smaxv = 0.0;
      for (jj = 0; jj < intsize; jj++)
        {
        if (sh.gain >= 0)
          tmaxv = ((float) (abs (int_buffer[jj])))*((float)(1 << sh.gain));
        else
          tmaxv = ((float) (abs (int_buffer[jj])));
        smaxv = max (smaxv, tmaxv);
        }
      smaxv1 = (double) smaxv;
      tmaxv1 = 2.0;
      rmaxv1 = log (smaxv1);
      rmaxv2 = log (tmaxv1);
      if ( sh.gain >= 0 )
        igain1 = (int) (tmaxv1 + rmaxv1/rmaxv2);
      else
        igain1 = 32 + sh.gain;

/* find the interferogram peak location and put into header */
      schead[1] = (float) sh.peak_location;

/*-----------------------------------------------------------------*/
/* move the data into the spectracalc header */

/*    set the data header init data bytes */
      icount2 = 1;
      memcpy (&ah.ivn, &ivn, icount2);
      memcpy (&ah.ihex, &ihex, icount2);

/*    set the gain in the header */
      ah.igain = igain1;

/*    set the number of points */
      ah.npts = numpts;

/*    set the starting point for data */
      ah.xbeg = xstart;

/*    set the ending point for data */
      ah.xend = xstop;
      memcpy( &ah.ixtype, &ixtype, icount2);
```

```
    memcpy( &ah.iytype, &iytype, icount2);

/*  insert the correct year into the header */
    icount2 = 4;
    memset(atime, 0, icount2);
    icount2 = 2;
    memcpy ( &atime, &(gh.date + 6), icount2);
    ah.iyear = 1900 + atoi (atime);

/*  insert the correct month into the header */
    icount2 = 4;
    memset(atime, 0, icount2);
    icount2 = 2;
    memcpy ( &atime, &gh.date, icount2);
    itemp = atoi (atime);
    memcpy ( &imonth, &itemp, icount2);
    icount2 = 1;
    memcpy( &ah.imonth, &imonth, icount2);
    icount2 = 4;

/*  insert the correct day into the header */
    memset(atime, 0, icount2);
    icount2 = 2;
    memcpy ( &atime, &(gh.date+3), icount2);
    itemp = atoi (atime);
    memcpy ( &iday, &itemp, icount2);
    icount2 = 1;
    memcpy( &ah.iday, &iday, icount2);

/*  insert the correct hour into the header */
    icount2 = 4;
    memset (atime, 0, icount2);
    icount2 = 2;
    memcpy ( &atime, &sh.scan_time, icount2);
    itemp = atoi (atime);
    memcpy ( &ihour, &itemp, icount2);
    icount2 = 1;
    memcpy ( &ah.ihour, &ihour, icount2);

/*  insert the correct minute into the header */
    icount2 = 4;
    memset(atime, 0, icount2);
    icount2 = 2;
    memcpy ( &atime, &(sh.scan_time+3), icount2);
    itemp = atoi (atime);
    memcpy (&iminute, &itemp, icount2);
    icount2 = 1;
    memcpy ( &ah.iminute, &iminute, icount2);

/*  insert the resolution information into the header */
    sprintf (iresol,"%.0f", gh.resolution);
    icount2 = 8;
```

```
     memcpy (&ah.iresol, &iresol, icount2);

/*   insert the peak maximum and other information into the header */
     ah.schead1 = schead[1];
     ah.schead2 = 0.0;
     ah.schead3 = 0.0;
     ah.schead4 = 0.0;
     ah.schead5 = 0.0;
     ah.schead6 = 0.0;
     ah.schead7 = 0.0;
     ah.schead8 = 0.0;

/*   insert user comments into the header */
     icount2 = 64;
     memcpy (&ah.scomm1, &gh.comm1, icount2);
     memcpy (&ah.scomm2, &gh.comm2, icount2);
     memcpy (&ah.scomm3, &gh.comm3, icount2);

/*------------------------------------------------------------------*/

/* write the SpectraCalc 256 byte header to disk */
     if (write (sptr, &ah, SCALC) != SCALC)
       {
        printf("\nERROR - can not write SpectraCalc header record\n");
        exit (10);
       }


/* convert the data to two's complement data format of SpectraCalc */
/* multiply by the gain from the binary interferogram file */
     itempl = 1;
     rmaxv1 = (double) ( itempl << ( 32 - igain1 ));
     for (jj = 0; jj < intsize; jj++)
       {
        if (sh.gain >= 0)
          rmaxv2=rmaxv1*(double)int_buffer[jj]*(double)(1 << sh.gain);
        else
          rmaxv2=(double)int_buffer[jj];
        intl_buffer[jj] = (long) rmaxv2;
       }

/* copy the long integer bytes to a short integer words */
     icount2 = 4*intsize;
     memcpy (&int_buffer, &intl_buffer, icount2);

/* reverse the integer words for the SpectraCalc 32 bit word format */
     for (j = 0; j < 2*intsize; j+=2)
      {
       itemp = int_buffer[j+1];
       int_buffer[j+1] = int_buffer[j];
       int_buffer[j] = itemp;
      }
```

```
/* write the buffer to disk */
     if (write(sptr, int_buffer, 4*intsize) != 4*intsize)
      {
       printf("\nERROR - Unable to write interferogram to data to disk.\n");
       exit(9);
      }
     }
/***************************************************************************/
/* close file and loop to get another interferogram */
    close (sptr);
    }

/* when finished... close the file and update the user */
   printf("\n\nTotal Number of Interferograms Converted ===> %4d\n",
           istop-istart+1);
   close(fp2);

  }
```

APPENDIX G

DATA COLLECTION PROGRAM (MIDCOLV)

```
/************************ program MIDCOL *********************************/
/*
 program MIDCOL                                       Version 4.0

   This program is used to read interferogram data, display,
   interferogram data, and Fourier transform the data  for
   display.  This program will be used for data collection
   for the Midac interferometer.

   author: Bob Kroutil, Mike Housky

   date: August 1992

   routines called:
      plotr     - plots an interferogram or spectrum
      logoega   - prints the CRDEC logo
      draw_axis - draws the axis for the plots for either interferogram
                  or spectra
      cmpfft    - computes the fast Fourier transform
      normal    - normalizes the spectrum
      MidAqInit - initialize the Midac interferometer
      MidAqStartScan - set up scanning for Midac
      Microsoft C graphics routines

-----------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <graph.h>
#include <math.h>
#include <string.h>
#include <time.h>

#include <stddef.h>/* Standard ANSI headers*/
#include <conio.h>/* MSC-specific headers*/
#include <malloc.h>
#include <dos.h>

#include "headers.def"  /* interferogram header information */
#include "middef.h"    /* MIDAC-specific headers*/
#include "menu.h"       /* menu display information      */
#include "exmidcol.def" /* external array definitions */

/* ---------------------------------------------------------------------- */
/*               Local definitions:                                       */
/* ---------------------------------------------------------------------- */

/* MSC7/MSC6 Portability:                                                 */

#ifdef MSC_VER
#if MSC_VER >= 700
```

```
#define outp _outp
#define inp  _inp
#endif
#endif

#define TIMEOUT 30.0            /* DMA Completion timeout, in seconds   */

/* Defaults for MidAqInit:                                              */

#define DMA              1      /* Default DMA channel                  */
#define DMAPAGE          0x83   /* DMA page register port for default   */
                /*   channel                           */
#define IRQ              2      /* Default IRQ channel                  */
#define GAIN             0      /* Default signal gain level (0-7)      */
#define BUFPTS           16384  /* Default DMA buffer size in data      */
                /*   points                            */
#define MAXDMA           0xFF80 /* Maximum DMA buffer size in bytes     */

            /* Note: MAXDMA must be less than the "ideal" limit of  */
            /* 64K for the GetDmaBuffer function to work properly.  */

/*
            System board (PC/AT) I/O definitions:
*/

#define SYS_DMA1         0x00   /* Base of byte DMA controller          */

/* These ports are channel-independent:                                 */

#define DMA_STAT  (SYS_DMA1+ 8) /* (R) Status register                  */
#define DMA_CMD   (SYS_DMA1+ 8) /* (W) Command register                 */
#define DMA_REQ   (SYS_DMA1+ 9) /* (W) Request register                 */
#define DMA_WSMR  (SYS_DMA1+10) /* (W) Write single mask register       */
#define DMA_MODE  (SYS_DMA1+11) /* (W) Mode register                    */
#define DMA_CLRF  (SYS_DMA1+12) /* (W) Clear byte pointer flip-flop     */
#define DMA_TEMP  (SYS_DMA1+13) /* (R) Temporary register               */
#define DMA_MCLR  (SYS_DMA1+13) /* (W) Master Clear                     */
#define DMA_CMSK  (SYS_DMA1+14) /* (W) Clear mask register              */
#define DMA_WAMR  (SYS_DMA1+15) /* (W) Write all mask register bits     */

/* These occur 4 times, once for each channel. Add 2*(channel number)   */
/* to get true port address:                                            */

#define DMA_ADDR (SYS_DMA1+ 0)  /* (R/W) Base or current address        */
#define DMA_CTR  (SYS_DMA1+ 1)  /* (R/W) Base or current word count     */

#define SYS_PIC1         0x20   /* Base of primary interrupt controller */
#define PIC1_CMD  (SYS_PIC1+0)  /* (W) Command register (OCW2/OCW3)     */
#define PIC1_STAT (SYS_PIC1+0)  /* (R) Status register (ISR or IRR)     */
#define PIC1_MASK (SYS_PIC1+1)  /* (R/W) Interrupt mask register        */

#define SYS_PIC2         0xA0   /* Base of secondary int. controller    */
```

```
#define PIC2_CMD  (SYS_PIC2+0)  /* (W) Command register (OCW2/OCW3)      */
#define PIC2_STAT (SYS_PIC2+0)  /* (R) Status register (ISR or IRR)      */
#define PIC2_MASK (SYS_PIC2+1)  /* (R/W) Interrupt mask register         */

#define PICC_EOI        0x20    /* OCW2 (nonspecific) End-Of-Interrupt   */
                /*   command                             */

/*
            Local Macros:
*/

#define PtrToLong(p) (((long)FP_SEG(p) << 4) + (long)FP_OFF(p))
                /* Macro to convert far pointer to      */
                /*   20-bit absolute address            */

#define DisableDma(ch) outp(DMA_WSMR, (ch)+4)   /* Disable DMA channel  */
#define EnableDma(ch)  outp(DMA_WSMR, (ch))     /* Enable DMA channel   */

/* Input and output from read-only command port, a shadow copy of the   */
/* port value is kept in MidGbl.CmpPort:                                 */

#define CmdIn()    (MidGbl.CmdPort)
#define CmdOut(val) (outp(MID_CMD, MidGbl.CmdPort = (int)(val)), \
                     outp(MID_CMD, MidGbl.CmdPort))

/* ------------------------------------------------------------------ */
/*               Global variables:                                     */
/* ------------------------------------------------------------------ */

MidAqGlobalType near MidGbl;    /* Global paramater/context variables   */

static int near DmaPageTable[8] = /* Table of DMA page register ports   */
            { 0x87, 0x83, 0x81, 0x82, -1, 0x8B, 0x89, 0x8A };

/* The following global parameters are the following:
      GH_LIMIT = the number of bytes in the global interferogram header
      SH_LIMIT = the number of points in the subfile interferogram header
      FEND     = the key code to exit the program
      FRIGHT   = the key code to expand the interferogram display
      FHOME    = the key code to reset the interferogram display
      FLEFT    = the key code to compress the interferogram display
      FINT     = the key code to display interferograms
      FSPEC    = the key code to display spectra
      FSCOL    = the key code to collect interferograms to disk
      FDIFF    = the key code to display a difference spectrum
      FBACK    = the key code to calculate a background spectrum
      FSEL5    = the key code to subtract the disk file f5.fsp
      FSEL6    = the key code to subtract the disk file f6.fsp
      FSEL7    = the key code to subtract the disk file f7.fsp
      FSEL8    = the key code to subtract the disk file f8.fsp
      ROLLL    = the key code to roll the display data to the left
      ROLLR    = the key code to roll the display data to the right
```

```
     PMODE    = the file read/write attributes
*/

#define GH_LENGTH 512
#define SH_LENGTH 64
#define FEND 79
#define FRIGHT 68
#define FHOME 71
#define FLEFT 67
#define FINT 59
#define FSEL5 63
#define FSEL6 64
#define FSEL7 65
#define FSEL8 66
#define FSPEC 60
#define FSCOL 61
#define FDIFF 62
#define ROLLL 75
#define ROLLR 77
#define PMODE 0644

/* function prototype for screen display */
ITEM mnuMain1[] =
 {
    { 0, "Laboratory"        },
    { 0, "Stationary-Ground"},
    { 0, "Mobile-Ground"     },
    { 0, "Hover-Air"         },
    { 0, "Flight-Air"        },
    { 0, ""                  }
 };
ITEM mnuMain2[] =
 {
    { 0, "A=1024 points"     },
    { 0, "B=2048 points"     },
    { 0, "C=4096 points"     },
    { 0, "D=8192 points"     },
    { 0, ""                  }
 };
ITEM mnuMain3[] =
 {
    { 0, "A=M2 jumper"       },
    { 0, "B=M1 jumper"       },
    { 0, "C=L1 jumper"       },
    { 0, "D=2L jumper"       },
    { 0, ""                  }
 };
ITEM mnuMain4[] =
 {
    { 0, "Yes"               },
    { 0, "No"                },
    { 0, ""                  }
```

```
};

main(argc, argv)
int argc;
char *argv[];
{
/* The following parameters are:
    raw_buf          - the interferogram buffer (real values)
    spc_buf          - the complex interferogram buffer, also used as a
                       work array
    pi               - value of the constant pi
    scan             - the scan number
    index            - an indexing variable
    fp2              - file open variables
    lpoints          - number of points in interferogram to display
    spoints          - number of points in the spectrum
    imode            - 0=display interferogram, 1=display spectrum
    inode            - set data collect switch
    loop             - graphics display page
    wscan            - scan number writing to disk
    ch               - used for an input
    bkgr             - the collect background flag
    ispts            - starting spectral plotting point for difference
                       spectrum
    iendp            - ending spectral plotting point for difference
                       spectrum
    lastpeak         - last array position of interferogram center burst
    extp             - the input extension filename
    drivep           - the input drive filename
    dirp             - the input directory filename
    icount2          - the index for number of bytes to copy
    outname          - the global header filename
    dirname          - the input filename to store to disk
    idate            - the array to hold the date
    itime            - the array to hold the time
    res              - the instrument resolution
    coll             - data collection mode
    itype            - integer data type
    speed            - interferometer scan speed
    mirror           - interferometer mirror movement
    sample           - spectral wavenumber sampling interval
    startf           - starting wavenumber
    stopf            - ending wavenumber
    mxwav            - maximum wavenumber frequency that can be sampled
    zcross           - number of zero crossings per sampled point
    temp             - ambient temperature
    barp             - barometric pressure
    humid            - relative humidity
    wind             - wind speed
    windd            - wind direction
    sendir           - sensor pointing direction
    precc            - precipitation code
```

```
    sensid            - array for sensor name
    opernam           - array for operators name
    global_header,gh  - the global header structure
    scan_header,sh    - the subfile header structure
    igain             - the A/D gain of the interferometer (0 - 7)
    limit             - the number of interferogram points to collect
    slimit            - the number of spectral points
    plimit            - the size of the interferogram array
    mdist             - interferometer scan length in centimeters
*/
  int MidAqInit(), MidAqSetGain(), Menu();
  void MidAqStartScan();
  int wrtint();
  void dispint(), dispspec(), diffspc(), logoega(), getspc();
  int inode, wscan, bkgr, spoints, jndex = 1;
  int scan, index, imode, loop=0, lastpeak, istps, iendp, ichng;
  char ch, buffer[4], outname[10], dirname[40], idate[10], itime[10];
  double res, mirror, speed, sample, startf, stopf, barp, mxwav;
  char comm1[64], comm2[64], comm3[64], comm4[64];
  int coll, itype, temp, humid, wind, windd, sendir, precc, ierr;
  int fp2, burst, zcross, maxscan, igain, plimit, slimit, limit;
  int iMainCur = 0, iMainCur1 = 0, iMainCur2 = 0, iMainCur3 = 0;
  int iCur = 0, iCur1 = 0, iCur2 = 0, iCur3 = 0, isample;
  int rowMid = 5, rowMid1 = 15, colMid = 15, colMid1 = 50;
  char sensid[20], opernam[10], extp[4], drivep[10], dirp[10], buf1[80];
  float pi, mdist;
  extern float raw_buf[], spc_buf[], spc_bak[];
  size_t hdcl, icount2=20;
  unsigned long t0,t1;
  struct global_header gh;
  struct scan_header sh;

/* set the maximum number of scans to collect by an input switch */
   if (argc == 2)
     maxscan = 550;
   else
      maxscan = 3000;

/* ask the user to input an output data collection filename */
    _clearscreen (_GCLEARSCREEN);
    printf("\nMIDCOL  -    Midac remote sensing data collection program
Version 4.0\n");
    printf("\nThe program switch is set to collect up to %d interferograms to
disk.\n",maxscan);
    printf("\nInput the data filename to store to disk: ");
    scanf ("%s",dirname);
    hdcl = 10;
    memset (&outname,32,hdcl);
    _splitpath (dirname, drivep, dirp, outname, extp);
    strupr (outname);

/* initialize the input buffers */
```

```
    hdcl =64;
    memset(&comm1,32,hdcl);
    memset(&comm2,32,hdcl);
    memset(&comm3,32,hdcl);
    memset(&comm4,32,hdcl);
    printf ("\nInput four lines for comments:\n");
    gets (comm1);
    printf (">>");
    gets (comm1);
    printf (">>");
    gets (comm2);
    printf (">>");
    gets (comm3);
    printf (">>");
    gets (comm4);

/* select the interferometer parameters from the display screen */
   _setvideomode(_DEFAULTMODE);
   _setbkcolor( (long)_TBLUE);
relook:
   _clearscreen( _GCLEARSCREEN);
   _settextposition (2,6);
   _outtext("Data collection mode ?");
   iMainCur = Menu( rowMid, colMid, mnuMain1, iCur);
   _settextposition (2,41);
   _outtext("Number of points to collect ?");
   iMainCur1 = Menu( rowMid, colMid1, mnuMain2, iCur1);
   _settextposition (12,3);
   _outtext("Midac sampling jumper setting ?");
   iMainCur2 = Menu( rowMid1, colMid, mnuMain3, iCur2);

/* set the input user parameters for the interferometer */
  switch (iMainCur)
    {
     case 0:
      coll = 0;
      break;
     case 1:
      coll = 1;
      break;
     case 2:
      coll = 3;
      break;
     case 3:
      coll = 4;
      break;
   }

  switch (iMainCur1)
   {
    case 0:
      limit = 1024;
```

```
      break;
    case 1:
      limit = 2048;
      break;
    case 2:
      limit = 4096;
      break;
    case 3:
      limit = 8192;
      break;
  }
  plimit = limit;
  slimit = 1 + (limit/2);

  switch (iMainCur2)
    {
     case 0:
      zcross = 800;
      break;
     case 1:
      zcross = 400;
      break;
     case 2:
      zcross = 200;
      break;
     case 3:
      zcross = 100;
      break;
    }

/* find the sampling parameters for the interferogram header */
  startf = 0.0;                           /* starting wavenumber */
  mxwav = 15798.0;                        /* laser wavenumber */
  stopf = mxwav /(float)(zcross/100);     /* ending wavenumber */
  sample = stopf/((float)(limit/2));      /* sampling wavenumber */
  isample = (int)(2*sample+1);
  res = (float)(isample);                 /* instrument resolution */
  mdist = ((float)(limit*zcross))/(mxwav*2.0);
                             /* interferometer scan length in centimeters */

/* notify the user of the selected interferometer parameters */
  _settextposition (20,20);
  _outtext("starting wavenumber=");
  _settextposition (20,40);
  sprintf(buf1,"%10.4f",startf);
  _outtext(buf1);
  _settextposition (21,20);
  _outtext("ending wavenumber=");
  sprintf(buf1,"%10.4f",stopf);
  _settextposition (21,40);
  _outtext(buf1);
  _settextposition (22,20);
```

```
  _outtext("sampling wavenumber=");
  sprintf(bufl,"%10.4f",sample);
  _settextposition (22,40);
  _outtext(bufl);
  _settextposition (23,20);
  _outtext("resolution=");
  sprintf(bufl,"%10.0f",res);
  _settextposition (23,40);
  _outtext(bufl);
  _settextposition (24,20);
  _outtext("zero crossing sampling=");
  _settextposition (24,49);
  isample = zcross/100;
  sprintf(bufl,"%d",isample);
  _outtext(bufl);

/* ask the user if all of the input is OK */
  _settextposition (12,43);
  _outtext("All answers correct ?");
  iMainCur3 = Menu( rowMidl, colMidl, mnuMain4, iCur3);
  switch (iMainCur3)
   {
    case 0:
     break;
    case 1:
     goto relook;
     break;
   }

 /* set up the graphics mode and clear screen */
   _setvideomode (_ERESCOLOR);
   _displaycursor( _GCURSOROFF);
   _setbkcolor (_BLUE);
   _settextposition (13, 20);
   _outtext ("Please Wait -- Initializing Interferometer");

/*==============create the global header================================*/
  /* create a new global header */

/* clear the global header buffers with blanks */
  hdcl=512;
  memset (&gh,32,hdcl);
  hdcl = 64;

/* initialize the default global header data parameters */
  itype = 1;                /* integer data type */
  temp = 0;                 /* ambient temperature */
  barp = 0.0;               /* barometric pressure */
  humid = 0;                /* relative humidity */
  wind = 0;                 /* wind speed */
  windd = 0;                /* wind direction */
  sendir = 0;               /* sensor direction */
```

```
  precc = 0;                   /* precipitation code */
  strcpy (sensid,"MIDAC unit #120     "); /* set the sensor name */
  strcpy (opernam,"          ");       /* blank out the operators name */

/*---------------------------------------------------------------------*/

/* stuff in the integer and double header information into
   the correct locations */

  gh.collect_mode = coll;
  gh.integer_type = itype;
  gh.scan_size = limit;
  gh.resolution = res;
  gh.sample_freq = sample;
  gh.start_freq = startf;
  gh.stop_freq = stopf;
  gh.max_wav = mxwav;
  gh.zercross = zcross;
  gh.ambient_temp = temp;
  gh.bar_pressure = barp;
  gh.humidity = humid;
  gh.wind_speed = wind;
  gh.wind_direction = windd;
  gh.sensor_direction = sendir;
  gh.precip_code = precc;

/* copy the sensor id */
  icount2 = 20;
  memcpy (&gh.sensor_id, &sensid, icount2);

/* copy the comment field */
  icount2=64;
  memcpy (&gh.comm1, &comm1, icount2);
  memcpy (&gh.comm2, &comm2, icount2);
  memcpy (&gh.comm3, &comm3, icount2);
  memcpy (&gh.comm4, &comm4, icount2);

/* find the starting date and time */
  _strtime (itime);
  icount2 = 10;
  memcpy (&gh.start_time, &itime, icount2);
  _strdate (idate);
  memcpy (&gh.date, &idate, icount2);

/* input the operators name */
  memcpy (&gh.operator, &opernam, icount2);

/* input the filename into the header */
  memcpy (&gh.filename, &outname, icount2);

  if (fp2 = creat (dirname, PMODE) < 0 )
    {
```

```
      _setvideomode (_DEFAULTMODE);
      printf ("\n\"MIDCOL\" is unable to create %s\n",dirname);
      exit(2);
     }
  if ((fp2 = open (dirname, O_WRONLY|O_BINARY)) < 0)
     {
       _setvideomode (_DEFAULTMODE);
       printf ("\n\"MIDCOL\" is unable to open %s\n",dirname);
       exit(2);
      }
  /* write the global header information */
    write (fp2, &gh, GH_LENGTH);

  /* set the parameter values for data collection */
  pi=4.*atan(1.);  /* the value of pi */
  imode = 0;        /* 0=display interferogram ; 1=display spectrum */
  istps = 1;        /* the starting point to display */
  iendp = 400;      /* the ending point to display */
  ichng = 50;       /* the display number of points to roll screen */
  spoints = limit; /* set the maximum point number to roll screen */
  wscan = 1;        /* initialize number of scans written to disk */
  inode = 0;        /* determines status of disk file */
  bkgr = 1;         /* set the background flag to collect */
  scan = -1;        /* initialize the scan data collection value */
  igain = -1;       /* have the gain initialize to initial value */

  /* This is the main loop for data collection to proceed */

 /* initialize the interferometer with scanning parameters */
    index = MidAqInit( -1, -1, igain, plimit);
    if (index)
    {
   _setvideomode (_DEFAULTMODE);
           printf("Error: MidAqInit returned %d\n", index);
           exit (2);
    }
/*    printf("MidCol initialized:\n");
    printf("  DMA Buffer at %Fp = %061X\n", MidGbl.DmaBuffer,
                PtrToLong(MidGbl.DmaBuffer)); */

/************************************************************************/
/* check the scan rate and store value in the header buffer */
   t0 = (unsigned long)clock();
   MidAqStartScan();
   while (!MidGbl.DmaDone)
   {
    t1 = (unsigned long) clock();
    if ((t1-t0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
    {
     _setvideomode (_DEFAULTMODE);
     printf("===> ERROR - no signal from interferometer <===");
     exit (2);
```

```
   }
  }
  MidGbl.DmaActive = 0;
  speed = 1.0/((float)(t1-t0)/(float)CLOCKS_PER_SEC);
  mirror = mdist * speed;
  gh.scan_speed = speed;
  gh.mirror_velocity = mirror;
/***********************************************************************/
/***********************************************************************/
 /* check the instrument gain -- if too low, then increase gain
                              if too high, then decrease gain */
/*    igain++;
   MidAqStartScan();
   t0 = (unsigned long)clock();
   while (!MidGbl.DmaDone)
    {
     t1 = (unsigned long)clock();
     if ((t1-t0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
       {
        _setvideomode (_DEFAULTMODE);
        printf("Error: Timeout on DMA completion\n");
        exit (2);
       }
    }
   MidGbl.DmaActive = 0;
   for (index=0; index < limit-1; index++)
       raw_buf[index+1] = (float) MidGbl.DmaBuffer[index];

   burst = fburst(raw_buf,limit-1);
   while(fabs(raw_buf[burst]) <= 16384. && igain <= 7)
    {
     raw_buf[burst] *= 2.;
     igain +=1;
    }
     MidAqSetGain(igain);
     printf(".... setting the instrument A/D gain to = %d",igain);
     MidAqStartScan();
     t0 = (unsigned long)clock();
     while (!MidGbl.DmaDone)
       {
         t1 = (unsigned long)clock();
         if ((t1-t0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
           {
            _setvideomode (_DEFAULTMODE);
            printf("Error: Timeout on DMA completion\n");
            exit (2);
           }
       }
      MidGbl.DmaActive = 0; */
/***********************************************************************/

 /* loop to collect interferogram data */
```

```
tloop:
    scan++;

 /* read in the interferogram data from the interferometer */
  MidAqStartScan();
  t0 = (unsigned long) clock();
  while (!MidGbl.DmaDone)
  {
    t1 = (unsigned long) clock();
    if ((t1-t0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
    {
      _setvideomode (_DEFAULTMODE);
      printf("Error: Timeout on DMA completion\n");
      exit (2);
    }

/*------------------ select user mode ----------------------------*/
     if (kbhit() != 0)  /* check to see if a key was pressed */
       {
         ch=getch();
         if (ch == FEND)  /* exit program */
           {
             if (inode == 1) /* if writing to disk update global header */
              {
               lseek (fp2, OL, 0); /* rewind the file to write header */
               gh.stop_scan = wscan - 1;  /* insert the number of scans in
header */
               _strtime (itime);   /* input the ending time into header */
               memcpy (&gh.stop_time, &itime, icount2);
               write (fp2, &gh, GH_LENGTH);/* write global header */
               close (fp2);
               }
            _setvideomode(_DEFAULTMODE);
            exit(1);
            }
         if (ch == FRIGHT) /* expand screen display */
           {
           iendp = iendp - ichng;
           istps = istps + ichng;
           if (istps >= iendp)
             {
              istps = istps - ichng;
              iendp = iendp + ichng;
             }
           }
         if (ch == FLEFT)/* contract the screen display */
           {
           iendp = iendp + ichng;
           istps = istps - ichng;
           if (istps < 1 ) istps = 1;
           if (iendp > spoints) iendp = spoints;
           }
```

```
if (ch == ROLLR) /* roll the data to the right */
  {
  iendp = iendp - ichng;
  istps = istps - ichng;
  if (istps < 1 )
    {
     istps = 1;
     iendp = iendp + ichng;
    }
  }
if (ch == ROLLL) /* roll the data to the left */
  {
  iendp = iendp + ichng;
  istps = istps + ichng;
  if (iendp > spoints)
     {
      iendp = spoints;
      istps = spoints - ichng;
     }
  }
 if (ch == FINT)/* display interferogram */
  {
   imode=0;
   istps = 1;
   iendp = 400;
   spoints = limit;
   }
if (ch == FSPEC)/*  display spectrum */
  {
   imode=1;
   istps = 1;
   iendp = slimit - 1;
   spoints= iendp;
  }
 if (ch == FSCOL) /* set disk data collection turned on */
  {
    imode = 2;
    inode = 1;
  }
 if (ch == FDIFF) /* display the difference spectrum */
  {
    imode = 3;
    bkgr = 1;
    istps = (int)((float)slimit * 181. / 512.);
    iendp = (int)((float)slimit * 363. / 512.);
    spoints = slimit;
  }
  if (ch == FSEL5 || ch == FSEL6 || ch == FSEL7 || ch == FSEL8)
  {
   imode = 3;
   bkgr = 0;
   istps = (int)((float)slimit * 181. / 512.);
```

```
            iendp = (int)((float)slimit * 363. / 512.);
            spoints = slimit;
            getspc (spc_bak, spoints, ch);
           }
           if (ch >= FINT && ch < FLEFT)
             jndex = (int) ch - 58;
       }
 /* return to check the keyboard if the scan is not finished */
     }
/*---------------------------------------------------------------------*/
     MidGbl.DmaActive = 0;

  /* convert the integer array to an ungain ranged floating array */
  for (index = 0; index < limit; index++)
    raw_buf[index+1] = (float) MidGbl.DmaBuffer[index];

  raw_buf[0]=0.0;
  spc_buf[0]=0.0;

/* set up the graphics to plot */
  loop = loop ^ 1;
  _setactivepage(loop);
  _clearscreen(_GCLEARSCREEN);
  _setvieworg(0,0);
   logoega(2,12);
  _setvieworg(64,175);

/* do the correct math operation for each selection */
/* display the interferogram to the screen */
  if (imode == 0)
     dispint (raw_buf, istps, iendp, imode, scan, jndex);
/* display the spectrum to the screen */
  else if (imode == 1)
     dispspec (raw_buf, spc_buf, limit, istps, iendp, pi, imode, scan, sample,
               jndex);
  else if (imode == 2)
     {
/* exit data collection if too many interferograms have been collected */
     if (wscan > maxscan)
       {
        lseek (fp2, 0L, 0); /* rewind the file header */
        gh.stop_scan = wscan - 1; /* insert the number of scans in header */
        _strtime(itime); /* get the ending time to put into header */
        memcpy (&gh.stop_time, &itime, icount2);
        write (fp2, &gh, GH_LENGTH); /* write the global header */
        close(fp2);
        _setvideomode(_DEFAULTMODE);
        exit(2);
       }

/* write the interferogram to the disk */
     lastpeak=wrtint (raw_buf, limit, wscan, lastpeak, outname, dirname,
```

```
                        fp2);
     wscan++;
     }
  else
/* write the difference spectrum to the screen */
     {
     diffspc (raw_buf, spc_buf, spc_bak, pi, bkgr, imode, istps,
              iendp, scan, limit, sample, jndex);
              bkgr=0;
     }

/* loop to get more data */

  _setvisualpage(loop);
  goto tloop;

}
/************************ end of program MIDCOL ********************/
/************************function dispint ***************************/
/* DISPINT

  This routine will display the interferogram on the screen for the
  real-time data collect option

  routines called:
  draw_axis    -  draw an axis to the screen
  plotr        -  plot the interferogram on the screen

---------------------------------------------------------------------*/
void dispint (raw_buf, istps, iendp, imode, scan, jndex)
/* The following global parameters are :
   raw_buf  - the interferogram data points to display
   imode    - the plotting mode to display 0=interferogram display
   istps    - the starting point to display
   iendp    - the ending point to display
   scan     - the scan number of the interferogram
   jndex    - the menu number to display on the screen
*/
float raw_buf[];
int istps, iendp, imode, scan, jndex;

 {
  void draw_axis(), plotr();
  int i;
  long int max_val=0, min_val=0, pktopk;
  char buffer[5];

/* find the peak to peak value of the interferogram */
  for (i = istps; i < iendp; i++)
   {
    max_val = max (MidGbl.DmaBuffer[i],max_val);
    min_val = min (MidGbl.DmaBuffer[i],min_val);
```

```
   }
   pktopk = max_val - min_val;

/* plot the interferogram data to the screen */
  draw_axis (scan,imode);
  plotr (raw_buf, istps, iendp, imode);
  _settextposition ( 2, 54);
  _outtext ("peak-to-peak = ");
  _settextposition ( 2, 70);
  sprintf (buffer,"%5ld",pktopk);
  _outtext(buffer);
  _settextposition (3, 2);
  sprintf (buffer,"%5d",max_val);
  _outtext(buffer);
  _settextposition (23, 2);
  sprintf (buffer,"%5d",min_val);
  _outtext(buffer);
  _settextposition (24, 10);
  sprintf (buffer,"%5d",istps);
  _outtext(buffer);
  _settextposition (24,70);
  sprintf (buffer,"%5d",iendp);
  _outtext(buffer);
  _settextposition (1,2);
  _outtext("F");
  sprintf (buffer," %ld",jndex);
  _outtext (buffer);
 }
/*************************end of dispint ******************************/
/************************function dispspec ****************************/
/* DISPSPEC

   This is the spectral display routine.  This routine will
Fourier transform and display each collected interferogram.

routines called:
cmpfft     -  Fourier transform
plotr      -  plot spectrum to screen
draw_axis  -  draw the axis to the screen

-----------------------------------------------------------------*/
void dispspec (raw_buf, spc_buf, limit, istps, iendp, pi, imode, scan, sample,
               jndex)
/* The following global variables are:
  raw_buf - the collected interferogram buffer
  spc_buf - the fourier transformed spectral buffer
  limit   - the number of points to transform
  istps   - the starting point to display
  iendp   - the ending point to display
  pi      - the value of PI
  imode   - the display mode; 1 = spectral buffer
  scan    - the scan number to display
```

```
  sample  - the sampling point spacing in wavenumbers
  jndex   - the menu option to display on the screen
*/
float raw_buf[], spc_buf[], pi, sample;
int istps, iendp, imode, scan, limit, jndex;
  {
   void cmpfft(), plotr(), draw_axis();
   float minx_val, maxx_val, miny_val = 0.0, maxy_val = 0.0;
   int i;
   char buffer[6];

/* do the fourier transform */
   cmpfft (raw_buf, spc_buf, limit, pi);

/* find the maximum and minimum values for the plotted spectrum */
   minx_val = sample * (istps-1);
   maxx_val = sample * iendp;
   for (i= istps; i < iendp; i++)
     maxy_val = max (raw_buf[i], maxy_val);

/* plot the spectrum data to the screen */
   draw_axis (scan,imode);
   plotr (raw_buf, istps, iendp, imode);
   _settextposition ( 3, 1);
   sprintf (buffer,"%6.0f",maxy_val);
   _outtext (buffer);
   _settextposition ( 23, 1);
   sprintf (buffer,"%6.0f",miny_val);
   _outtext (buffer);
   _settextposition ( 25, 5);
   sprintf (buffer,"%6.0f",minx_val);
   _outtext (buffer);
   _settextposition ( 25, 70);
   sprintf (buffer,"%6.0f",maxx_val);
   _outtext (buffer);
   _settextposition (1,2);
   _outtext("F");
   sprintf (buffer," %1d", jndex);
   _outtext (buffer);
   }
/*************************end of dispspec ****************************/
/*************************function wrtint ****************************/
/* WRTINT

   This routine will write an interferogram to the disk.

   routines called:
   errcod   -   find the interferogram error code
   fburst   -   find the interferogram centerburst

-------------------------------------------------------------------*/
int wrtint (raw_buf, limit, wscan, lastpeak, outname, dirname, fp2)
```

```
/* The following global parameters are:
 raw_buf - the interferogram collected on the Midac
 limit   - the number of points in the array buffer
 wscan   - the last interferogram nummber written to disk
 lastpeak- the last interferogram burst position
 outname - the header name to store
 dirname - the directory name to store to disk
 fp2     - file pointers for disk I/O
*/
int wscan, limit, lastpeak, fp2;
float raw_buf[];
char dirname[], outname[];

  {
   int errcod(), fburst();
   int burst, ercod;
   size_t hdcl=64, icount2=10;
   char itime[10], buffer[4];
   struct scan_header sh;
   struct global_header gh;

/* initialize the subfile header information */
   memset (&sh, 32, hdcl);   /* initialize the subfile header buffer */
   burst = fburst (raw_buf, limit); /* find the center burst */
   if (wscan == 1)
    lastpeak = burst;
   sh.scan_number = wscan;          /* insert the scan number */
   sh.peak_location = burst;        /* centerburst position */
   sh.gain = MidGbl.GainVal;        /* interferogram A/D gain */
   sh.coadd = 1;      /* set the number of coadded interferograms */
   ercod = errcod (raw_buf, limit, burst, lastpeak);
   sh.error = ercod;                /* interferogram error code */
   lastpeak = burst;                /* set the last peak position
                                       for the centerburst */

/* put the header name into the source filename field */
   memcpy (&sh.filename, &outname, icount2);

/* find the scan time to put into the header */
  _strtime (itime);
   memcpy (&sh.scan_time, &itime, icount2);

/* write the interferogram to disk */
   /* write the subfile header information */
   write (fp2, &sh, SH_LENGTH);
   /* write the interferogram data to disk */
   write (fp2, MidGbl.DmaBuffer, limit*2);

/* display the information the the screen */
  _settextposition( 12, 20);
  _outtext("COLLECTING INTERFEROGRAM DATA TO DISK");
  _settextposition( 14, 20);
```

```
  _outtext("filename = ");
  _settextposition( 14, 32);
  _outtext(dirname);
  _settextposition( 16, 20);
  _outtext("interferogram number = ");
  _settextposition( 16, 44);
  sprintf (buffer, "%04d", wscan);
  _outtext (buffer);
  _settextposition ( 18, 20);
  _outtext("error code = ");
  _settextposition ( 18, 33);
  sprintf (buffer, "%01d", ercod);
  _outtext (buffer);
  return(lastpeak);
  }
/*************************end of wrtint ********************************/
/************************function diffspc ****************************/
/* DIFFSPC

  This routine will display a difference spectrum to the screen.

  routines called:
  cmpfft      -   Fourier transform
  normal      -   normalize a spectral buffer
  plotr       -   plot a spectral buffer to the screen
  draw_axis   -   plot the axis labels to the screen

---------------------------------------------------------------------*/
void diffspc (raw_buf, spc_buf, spc_bak, pi, bkgr, imode, sstart,
              send, scan, limit, sample, jndex)
/* The following parameters are:
             raw_buf  -  real array of interferogram values
    spc_buf  -  real array of spectral values
    spc_bak  -  real array of spectral background values
    pi       -  the value of pi
    bkgr     -  the background computation switch
    imode    -  the data display mode
    sstart   -  the starting point to plot the difference spectrum
    send     -  the ending point to plot the difference spectrum
    limit    -  the interferogram array size
    sample   -  the sampling point increment (wavenumbers)
    jndex    -  the menu option number to display on the screen
*/

float raw_buf[], spc_buf[], spc_bak[], pi, sample;
int bkgr, imode, sstart, send, scan, limit, jndex;
  {
   void cmpfft(), normal(), plotr(), draw_axis();
   float minx_val, maxx_val, miny_val=0.0, maxy_val=0.0;
   int index;
   char buffer[6];
```

```
   if (bkgr == 1)
     {
      cmpfft (raw_buf, spc_buf, limit, pi);
/*     normal (raw_buf, spoints);   */
      for (index=1; index <= limit/2; index++)
        spc_bak[index-1] = raw_buf[index];
              }
   else
     {
      cmpfft (raw_buf, spc_buf, limit, pi);
/*     normal (raw_buf, spoints);  */
      for (index= sstart; index < send; index++)
       {
         raw_buf[index]=raw_buf[index]-spc_bak[index-1];
         miny_val = min (raw_buf[index], miny_val);
         maxy_val = max (raw_buf[index], maxy_val);
       }
      draw_axis( scan, imode);
      plotr (raw_buf, sstart, send, imode);

/* annotate the screen with the display ranges */
     minx_val = sample * (sstart-1);
     maxx_val = sample * send;
     _settextposition ( 3, 1);
     sprintf (buffer,"%6.0f",maxy_val);
     _outtext (buffer);
     _settextposition ( 23, 1);
     sprintf (buffer,"%6.0f",miny_val);
     _outtext (buffer);
     _settextposition ( 25, 5);
     sprintf (buffer,"%6.0f",minx_val);
     _outtext (buffer);
     _settextposition ( 25, 70);
     sprintf (buffer,"%6.0f",maxx_val);
     _outtext (buffer);
     _settextposition ( 1, 2);
     _outtext("F");
     sprintf (buffer," %ld", jndex);
     _outtext (buffer);
     }
   }
/*************************end of diffspc ******************************/
/************************function cmpfft *****************************/
/* CMPFFT

   This routine will Fourier transform an interferogram.  The program
   will rotate the interferogram and transform.  No phase correction
   or apodization is done.  This routine is to be only used for
   real-time display where phase and apodization functions are not
   absolutely required.  Do not use this routine for data analysis.

   routines called:
```

```
     rotate   - rotates an interferogram buffer
     burst    - finds the centerburst of an interferogram
     rfft     - calculates the Fourier transformation

-------------------------------------------------------------------*/
void cmpfft (raw_buf, spc_buf, ipoints, pi)
/*  The following global parameters are:
    raw_buf    - a work array used for transformation
    spc_buf    - an array containing the complex values of the transformation
    ipoints    - number of points in interferogram array
    pi         - value of pi
*/
float raw_buf[], spc_buf[], pi;
int ipoints;
{
/*  The following local parameters are:
    i,j,index  - indexing variables
    burst      - value containing the index of the interferogram centerburst
*/
  void rfft(),rotate();
  int fburst();
  int i, j, index, burst;

   for (i=1; i <= ipoints; i++)
    spc_buf[i] = raw_buf[i];

/* find the center burst of the interferogram */
/*  printf ("to burst\n");  */
/*  printf ("raw_buf[50]= %10.5f\n",raw_buf[50]); */
  burst=fburst(spc_buf,ipoints);
/*  printf ("after burst\n"); */

/* rotate the interferogram for the FFT */
/*  printf ("to rotate\n");  */
  rotate(burst, spc_buf, raw_buf, ipoints);
/*  printf ("after rotate\n");  */

/* Fourier transform the interferogram */
/*  printf ("to rfft\n");  */
  for (i=1, j=1; j <= ipoints; i+=2, j++)
    {
      spc_buf[i] = raw_buf[j];
      spc_buf[i+1] = 0.0;
/*      printf ("spc_buf[%04d]=%10.5f\n",i,spc_buf[i]);*/
/*      printf ("spc_buf[%04d]=%10.5f\n",i+1,spc_buf[i+1]);  */
    }

  rfft(spc_buf, ipoints, pi);
/*  printf ("after rfft\n"); */

/* compute the power spectrum */
/*  printf ("to power spectrum calculation\n"); */
```

```
   for ( i=1, j=0 ; i <= ipoints ; i+=2, j++)
   {
   raw_buf[j]= sqrt(spc_buf[i]*spc_buf[i]+spc_buf[i+1]*spc_buf[i+1]);
/*   printf ("raw_buf[%04d]=%10.5f\n",j,raw_buf[j]); */
   }
/*   printf ("after power spectrum calculation\n"); */
  }
/*********************** end of CMPFFT ******************************/
/*********************** function rfft ******************************/
/* RFFT

  This routine will compute the Fourier transform using the method
originally written by N. Brenner of Lincoln Laboratories

  routines called:
    NONE

-------------------------------------------------------------------*/
void rfft (spc_buf, ipoints, pi)
/*  The following global parameters are:
    spc_buf  - the interferogram values stored in complex form
    ipoints  - number of points in interferogram
    pi       - value of pi
*/
float spc_buf[], pi;
int ipoints;
  {
    int i, n, istep, j, mmax, m;
    float wsin, theta, tempr, tempi, wr, wi, wtemp, wpr, wpi;

    n= 2 * ipoints;
    j=1;
   /* bit reversal section */
    for (i=1; i <= n ; i+=2)
     {
       if (j > i)
        {
/* Note: several statements have been commented out for the case
         where input imaginary values are always zero.  If this is
         not true, then these statements must be used.
*/
          tempr = spc_buf[j];
/*          tempi = spc_buf[j+1];    */
          spc_buf[j] = spc_buf[i];
/*          spc_buf[j+1] = spc_buf[i+1];  */
          spc_buf[i] = tempr;
/*          spc_buf[i+1] = tempi;  */
        }
       m=n/2;
       while ( m >= 2 && j > m )
         {
          j=j-m;
```

```
            m=m/2;
          }
        j=j+m;
      }

   /* compute the butterflies */

      mmax=2;
      while ( n > mmax )
        {
        istep= 2 * mmax;
        theta = 2.0 * pi /(float)mmax;
        wsin = sin(0.5 * theta);
        wpr = -2.0*wsin*wsin;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;
        for (m=1; m <= mmax; m+=2)
          {
            for (i=m; i <= n; i=i+istep)
              {
                j=i+mmax;
                tempr = wr*spc_buf[j] - wi*spc_buf[j+1];
                tempi = wr*spc_buf[j+1] + wi*spc_buf[j];
                spc_buf[j] = spc_buf[i] - tempr;
                spc_buf[j+1] = spc_buf[i+1] - tempi;
                spc_buf[i] = spc_buf[i] + tempr;
                spc_buf[i+1] = spc_buf[i+1] + tempi;
              }
            wtemp = wr;
            wr = wr*wpr - wi*wpi + wr;
            wi = wi*wpr + wtemp*wpi + wi;
          }
          mmax=istep;
      }
  }
/*********************** end of RFFT ********************************/
/*********************** function fburst *****************************/
/* FBURST

    This routine will find the center burst of an interferogram array.
    The routine is a function call as the burst value is returned.

    routines called:
       NONE
-------------------------------------------------------------------*/
int fburst(raw_buf,ipoints)
/* The following global parameters are:
      raw_buf  - the interferogram array
      ipoints  - the number of points in interferogram
*/
float raw_buf[];
```

```
int ipoints;
 {
   int i,max_loc, min_loc;
   float max_val=0.0, min_val=0.0;

/*   printf ("ipoints in fburst= %04d\n",ipoints);
   printf ("raw_buf[50]= %10.5f\n",raw_buf[50]);*/
   for (i=1;i <= ipoints; i++)
    if (raw_buf[i] > max_val)
      {
        max_val=raw_buf[i];
        max_loc = i;
/*        printf("max_loc= %04d\n",max_loc);  */
      }
    else if (raw_buf[i] < min_val)
      {
        min_val = raw_buf[i];
        min_loc = i;
/*        printf ("min_loc= %04d\n",min_loc);  */
      }

     if (fabs((double) min_val) > max_val)
        return (min_loc);
   else
      return (max_loc);
  }
/*********************** end of FBURST ******************************/
/*********************** function normal *****************************/
/* NORMAL

   This routine is used to normalize the spectral buffer.

   routines called:
    NONE

-----------------------------------------------------------------*/
void normal (buffer, ipoints)
float buffer[];
{
  int index;
  float ssq = 0.0;

  for (index = 0; index < ipoints; index++)
    ssq += buffer[index] * buffer[index];

  if (ssq > 0.0)
    ssq = ipoints / sqrt (ssq);
  else
    ssq = 1.0;

  for (index = 0; index < ipoints; index++)
    buffer[index] *= ssq;
```

```
}
/************************ end of normal ********************************/
/*********************** function rotate ******************************/
/* ROTATE

   This routine will rotate an interferogram buffer.  The buffer will
   be rotated so that the center burst is in array position 1.

   routines called:
      NONE

------------------------------------------------------------------*/
void rotate (burst, raw_buf, spc_buf, ipoints)
/* The following parameters are:
    raw_buf  -  the input interferogram buffer
    spc_buf  -  the rotated interferogram buffer
    burst    -  the interferogram center burst array position
    ipoints  -  number of interferogram points is arrays
*/
float raw_buf[], spc_buf[];
int ipoints, burst;
 {
  int oindex, nindex;

  for (oindex=burst, nindex=1; oindex <= ipoints; oindex++, nindex++)
    spc_buf[nindex] = raw_buf[oindex];
/*  nindex-=1;      */
  for (oindex=1; oindex < burst; oindex++)
    {
     spc_buf[nindex] = raw_buf[oindex];
     nindex++;
    }
 }
/************************ end of ROTATE ********************************/
/*********************** function draw_axis ***************************/
/* DRAW_AXIS

   This routine will draw the axis for either an interferogram or
   spectrum display.

   routines called:
     Microsoft C graphics display routines

------------------------------------------------------------------*/
void draw_axis (scan,imode)
/* The following parameters are:
     scan  - the scan number
     imode - display mode type; 0=interferogram, 1=spectrum
*/
int scan, imode;
{
  int  i, ih;
```

```
char buffer[80];

if (imode == 1)
  ih = 150;
else
  ih = 0;

_moveto (0, ih+0);   /* Print the X axis */
_lineto (512, ih+0);
_moveto (0,150);   /* Print the Y axis */
_lineto (0,-150);

for(i = 0; i <= 512; i += 64)  /*Print the X axis tick marks */
  {
  _moveto(i, ih+5);
  _lineto(i, ih+0);
  }

for(i = 0; i <= 512; i += 32)
  {
  _moveto(i, ih+3);
  _lineto(i, ih+0);
  }

for(i = 0; i <= 512; i += 16)
  {
  _moveto(i, ih+2);
  _lineto(i, ih+0);
  }

/*  for(i = 150; i > -150; i -= 25)   Print the Y axis tick marks
    {
    _moveto(-4, i+1);
    _lineto(0, i+1);
    }  */

/* Label the axis */
_settextposition(25,36);          /*  X AXIS */
_outtext (" SCAN # ");
 sprintf(buffer,"%05d",scan);
_settextposition(25,45);
_outtext (buffer);
if (imode == 3)
  {
    _settextposition (25, 8);
    _outtext ("700");
    _settextposition (25, 70);
    _outtext ("1400");
  }

_settextposition(9,5);          /* Y AXIS */
_outtext ("A");
```

```
  _settextposition(10,5);
  _outtext ("/");
  _settextposition(11,5);
  _outtext ("D");
  _settextposition(13,5);
  _outtext ("u");
  _settextposition(14,5);
  _outtext ("n");
  _settextposition(15,5);
  _outtext ("i");
  _settextposition(16,5);
  _outtext ("t");
  _settextposition(17,5);
  _outtext ("s");

}
/**************************** end of DRAW_AXIS *******************************/
/**************************** function logoega *******************************/
/*  logoega is a function used to create the CBDA logo for EGA graphics.
    The funtion requires two parameters, the x and y coordinates for the
    first letter "C". If the logo coordinates are outside the exceptable
    range, no logo will be plotted.

   author: John Ditillo
   modified by: Bob Kroutil

           logoega is based on the "old" CRDEC logo routine
           written by John T. Ditillo

   date: October 1992  */

void logoega(y,x)
int y, x;

{
  int xp, yp;

  if (y<23 & y>1 & x<76 & x>2)
    {

    /* draw the logo */
    _settextposition(y,x);
    _outtext ("C");

    _settextposition(y+1,x-1);
    _outtext ("B D");

    _settextposition(y+2,x);
    _outtext ("A");

    /* Calculate first pixel location */
    yp = y * 14 - 16;
```

```
    xp = x * 8 - 5;

    /* first benzene */
    _moveto(xp,yp);
    _lineto(xp-8,yp+3);
    _lineto(xp-8,yp+13);
    _lineto(xp,yp+17);
    _lineto(xp+8,yp+13);
    _lineto(xp+8,yp+3);
    _lineto(xp,yp);

    /* second benzene */
    _moveto(xp-8,yp+13);
    _lineto(xp-16,yp+17);
    _lineto(xp-16,yp+27);
    _lineto(xp-8,yp+31);
    _lineto(xp,yp+27);
    _lineto(xp,yp+17);

    /* third benzene */
    _moveto(xp+8,yp+13);
    _lineto(xp+16,yp+17);
    _lineto(xp+16,yp+27);
    _lineto(xp+8,yp+31);
    _lineto(xp,yp+27);

    /* fourth benzene */
    _moveto(xp-8,yp+31);
    _lineto(xp-8,yp+42);
    _lineto(xp,yp+45);
    _lineto(xp+8,yp+42);
    _lineto(xp+8,yp+31);

  }

}
/***********************  end of LOGOEGA ****************************/
/***********************  function plotr ****************************/
/* PLOTR

   This routine is used to scale and display the interferogram or
   spectrum.

   routines called:
    Microsoft C graphics routines

--------------------------------------------------------------------*/
void plotr (buf, istps, iendp, imode)
/* The following parameters are:
      buf    - the array buffer to plot
      istps  - the starting point to display
      iendp  - the ending point to display
```

```
     imode  - the display mode (0=interferogram, 1=spectrum)
*/
float buf[];
int istps, iendp, imode;
{
  int index, x, y, ih, ip;
  float max, xscale, yscale;

  /* number of points to plot */
  ip = iendp - istps;

  /* find the largest value */
  for (index=istps, max=0.0; index < iendp; index++)
    {
    if ((fabs((double)buf[index])) > max)
      max = (float) (fabs((double)buf[index]));
    }

  /* Calculate the scaling factor */
  xscale = 512.0/ip;
  if (imode == 1)
    {
    yscale = 300.0/max;
    ih = 150;
    }
  else
    {
    yscale = 150.0/max;
    ih = 0;
    }

  /* plot the data */
  _moveto (0, (int) -(buf[istps] * yscale - ih));
  for (index=1; index < ip; index++)
    {
     x = (int) index * xscale;
     y = (int) -(buf[index+istps] * yscale - ih);
     _lineto (x,y);
    }

}
/**************************** end of PLOTR ****************************/
/********************** function getspc *******************************/
/* GETSPC

  This routine will get up to 4 black body spectra on the disk and
  read them into an array.  The stored spectra are SpectraCalc
  floating point binary format (FSP format - use the input and
  output commands in SpectraCalc).

  routines called:
   NONE
```

```
---------------------------------------------------------------------*/
void getspc (spc_bak, ipts, ch)
/* The following parameters are:
  spc_bak - the array that contains the stored disk spectral responses
  ipts    - the number of points in the array
  ch      - a flag to tell which spectrum file to read
*/

float spc_bak[];
int ipts;
char ch;
  {
   int fp3;
   float numpts, firstx, lastx, xunits, yunits, res;
   char afile[20];

/* load the black body spectra */

   if (ch == FSEL5)
     strcpy (afile,"f5.fsp");
   if (ch == FSEL6)
     strcpy (afile,"f6.fsp");
   if (ch == FSEL7)
     strcpy (afile,"f7.fsp");
   if (ch == FSEL8)
     strcpy (afile,"f8.fsp");

   if ((fp3 = open (afile,O_RDONLY|O_BINARY)) >= 0)
     {
      read (fp3, (char *) &numpts, 4);
      read (fp3, (char *) &firstx, 4);
      read (fp3, (char *) &lastx, 4);
      read (fp3, (char *) &xunits, 4);
      read (fp3, (char *) &yunits, 4);
      read (fp3, (char *) &res, 4);

      if ( read (fp3, spc_bak, 4 * ipts) != 4 * ipts)
        printf("\nUnable to read disk stored black body file.\n");
      close (fp3);
      }
    else
     {
      _settextposition (1,20);
      _outtext ("===> ERROR - disk file .fsp does not exist !!!! <===");
     }
  }
/********************* end of getspc *********************************/
/********************* function errcod *******************************/
/* ERRCOD

   This routine will find out if the data has an error.
```

```
   routines called:
     NONE

------------------------------------------------------------------*/
int errcod (raw_buf, ipoints, burst, lastpeak)
/* The following parameters are:
     raw_buf - the real valued buffer array to test
     ipoints  - number of points in array
     burst    - the array location of the center burst
     lastpeak - the last array location holding the previous center burst
*/

int burst, lastpeak, ipoints;
float raw_buf[];
   {
     int ercod;

     ercod = 0;

     if (ipoints < 1024)
       ercod = 1;
     if (fabs(raw_buf[burst]) >= 32767.)
       ercod = 2;
     if (lastpeak != burst)
       ercod = 3;
     if (burst > 500)
       ercod = 4;
     if (fabs(raw_buf[burst]) <= 8192.)
       ercod = 5;
/*     printf ("raw_data[%04d] = %05d",burst, raw_data[burst]);
       printf ("burst position = %04d", burst);  */

/* NOTE: error code for bit toggle not yet implemented */

      return (ercod);
    }
/*------------------------------------------------------------------- */
/*      in:     Allow port input during debug.                        */
/*              This is necessary for CV 4.00--the "I" command (port  */
/*              input is broken. The circumvention is to include a    */
/*              a global function such as in() below, trace at least  */
/*              as far as the main() function, then "?in(port)" or    */
/*              "?in(port),x" to read port contents.                  */
/* ------------------------------------------------------------------ */

int in( unsigned port )
{
    int i;
    i = inp(port);
    return i;

} /* in */
```

```
/* ---------------------------------------------------------------- */
/*      IoDelay:        I/O delay for IBM/AT and clones.              */
/*                                                                    */
/*      This dummy function is used to generate a few clocks of delay */
/*      between consecutive accesses to certain I/O ports. Basically, */
/*      the call/return sequence is more than enough. Assembler       */
/*      programs typically use a "JMP SHORT $+2" instruction, but     */
/*      the MSC7 inline assembler doesn't seem to handle the "$"      */
/*      token very well. The delay is necessary on IBM AT machines    */
/*      and true compatibles.                                         */
/*                                                                    */
/*      Needless to say, allowing this function to be inlined would   */
/*      be a bad idea...                                              */
/* ---------------------------------------------------------------- */

static void near IoDelay(void)
{
   ;
} /* IoDelay */

/* ---------------------------------------------------------------- */
/*      GetDmaBuffer:   Allocate a byte-DMA compatible buffer         */
/*                                                                    */
/*      A byte DMA buffer cannot cross a 64K-byte absolute address    */
/*      boundary.                                                     */
/*                                                                    */
/*      Returns pointer to buffer if successful, NULL otherwise.      */
/* ---------------------------------------------------------------- */

void far *GetDmaBuffer(long Size)
{
    #define MaxTries 16         /* Maximum attempts before failure     */

    void        far *failed[MaxTries],
                 far *try,
                 far *retry;
    unsigned    begoff, endoff;
    int         i, nfail=0;

    if (Size>MAXDMA || Size<=0) return NULL;

    for (;;)                            /* Repeat until explicit break: */
    {
            try = malloc((size_t)Size);
            if ( try==NULL ) break;

/* Test for 64K block wraparound:                                     */

            begoff = (FP_SEG(try) << 4) + FP_OFF(try);
            endoff = begoff + (unsigned)Size - 1;
            if (endoff >= begoff) break;    /* Success if all in 1 block    */
```

```
/* Current attempt crosses boundary, retry if failed list not full:     */

            if (nfail == MaxTries)
            {
                free(try);
                try = NULL;
                break;
            }

/* Resize current try to end on 64K absolute boundary and add it to     */
/* the failed list:                                                     */

            retry = realloc(try, 1+~begoff);
            if ( retry != NULL )
                try = retry;
            failed[nfail++] = try;
    }

/* Arrive here via explicit break. Free failed attempt pointers, if     */
/* any and exit. The try variable has been set to a pointer on success  */
/* or to NULL on error.                                                 */

    for( i=0; i<nfail; ++i )
    {
            free( failed[i] );
    }

    return try;

#undef MaxTries                            /* Undefine "local" macros     */

} /* GetDmaBuffer */


/* ------------------------------------------------------------------ */
/*     StartDma:       Start a DMA operation.                           */
/*                                                                      */
/*     This is a cut-down version to do input only, specifically        */
/*     using DMA info in MidGbl structure.                              */
/* ------------------------------------------------------------------ */

void StartDma(void)
{
    long        addr = PtrToLong(MidGbl.DmaBuffer);
    int         size = (int)MidGbl.DmaSize;
    unsigned    ch   = 2*MidGbl.DmaChannel;

    DisableDma(MidGbl.DmaChannel);
    IoDelay();                             /* Wait a few CPU clocks       */
    outp(DMA_MODE, 0x44+MidGbl.DmaChannel);
                /* DMA Mode: single-block,      */
                /*   increment address,         */
```

```
                /*   no autoinitialize,         */
                /*   "write transfer" -> cpu    */
    IoDelay();                          /* Wait a few CPU clocks        */

    outp(DMA_CLRF,0);                   /* Set to receive LSB first     */
    IoDelay();                          /* Wait a few CPU clocks        */

    outp(DMA_CTR+ch, (int)size);        /* Send byte count              */
    IoDelay();                          /* Wait a few CPU clocks        */
    outp(DMA_CTR+ch, (int)size >> 8);
    IoDelay();                          /* Wait a few CPU clocks        */

    outp(DMA_ADDR+ch, (int)addr);       /* Send address                 */
    IoDelay();                          /* Wait a few CPU clocks        */
    outp(DMA_ADDR+ch, (int)addr >> 8);
    IoDelay();                          /* Wait a few CPU clocks        */

    outp(MidGbl.DmaPageReg, (int)(addr>>16));
                /* Set page reg to top 8 bits   */
    IoDelay();                          /* Wait a few CPU clocks        */

    EnableDma(MidGbl.DmaChannel);       /* Finally, enable DMA          */

} /* StartDma */


/* ------------------------------------------------------------------ */
/*      SetIrqEnable:   Set/Reset IRQ enable status for specified      */
/*                      channel.                                       */
/*                                                                     */
/*      Please note that the sense of the "Enable" argument is a C-    */
/*      style boolean. Nonzero, or "true", enables the channel. This   */
/*      is opposite from the 8259 mask register, where a 1 disables    */
/*      the channel and 0 enables.                                     */
/* ------------------------------------------------------------------ */

void SetIrqEnable(
    int         IrqNumber,      /* Interrupt channel, 0-15              */
    int         Enable)         /* New enable status for this channel   */
                /*   0 = disable interrupts          */
                /*   nonzero = enable interrupts     */
{
    unsigned    port;
    int         mask, val;

    if (IrqNumber < 8)
    {
        port = PIC1_MASK;               /* Primary 8259 port            */
        mask = 1 << IrqNumber;
    }
    else
    {
```

```
        port = PIC2_MASK;                /* Secondary 8259 port          */
        mask = 1 << (IrqNumber-8);
    }

    val = inp(port) | mask;              /* Set to mask disable          */
    if (Enable) val -= mask;             /* Set to enable if requested   */
    outp(port, val);                     /* Update port                  */

} /* SetIrqEnable */


/* ------------------------------------------------------------------ */
/*      MidAqStartScan: Start new data collect operation              */
/*                                                                    */
/*      This is a skeleton of what is needed to begin a new data      */
/*      scan, or series of accumulated scans, on the Midac FT-IR.     */
/* ------------------------------------------------------------------ */

void MidAqStartScan(void)
{
    SetIrqEnable(MidGbl.IrqNum, 0);      /* Disable interrupt channel    */
    IoDelay();                           /* Wait a few CPU clocks        */
    DisableDma(MidGbl.DmaChannel);       /* Disable DMA channel          */
    IoDelay();                           /* Wait a few CPU clocks        */

    StartDma();                          /* Start DMA channel            */

    SetIrqEnable(MidGbl.IrqNum, 1);      /* Enable interrupt channel     */

/* Set gain and retrace interferometer:                                */

    CmdOut( MidGbl.GainPort | MIDC_EOS | MIDC_IRQ );
              /* Start IRQ clear pulse*/
    IoDelay();                                   /* Wait a few CPU clocks*/
    CmdOut( CmdIn() &~(MIDC_EOS + MIDC_IRQ) );   /* End IRQ clear pulse, */
              /* Start retrace pulse  */
    IoDelay();                                   /* Wait a few CPU clocks*/
    while (inp(MID_STAT) & MIDS_FLYBK);          /* Wait for turnaround  */
    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ));    /* End retrace pulse    */
    IoDelay();                                   /* Wait a few CPU clocks*/

    /* Note: May need to insert delay here, 10-20ms, to allow for      */
    /* hardware bug in Midac interface causing early DMA requests.     */
              _asm xor   cx,cx
        here:    _asm loop here

    MidGbl.DmaActive = 1;                /* Set global DMA status flags  */
    MidGbl.DmaDone = 0;

    CmdOut( CmdIn() | MIDC_DMA );        /* Enable DMA at interface      */

} /* MidAqStartScan */
```

```
/* ---------------------------------------------------------------------- */
/*      MidAqDmaDone:   Interrupt Handler for DMA completion              */
/*                                                                        */
/*      This version simply notes DMA completion, retraces the            */
/*      interferometer, and disables DMA at both the 8237 and at          */
/*      the Midac interface board.  This would be the natural place       */
/*      to insert co-add logic for averaging interferograms.              */
/* ---------------------------------------------------------------------- */

void _cdecl _interrupt far MidAqDmaDone(void)
{
    MidGbl.DmaDone = 1;                    /* Note DMA completion         */

    CmdOut( CmdIn() &~MIDC_DMA );          /* Disable DMA at interface    */
    DisableDma(MidGbl.DmaChannel);         /*  then disable channel       */
    IoDelay();                             /* Wait a few CPU clocks       */

/* Retrace interferometer:                                                */

    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ) );  /* Start IRQ clear pulse*/
    CmdOut( CmdIn() &~(MIDC_EOS + MIDC_IRQ) );  /* End IRQ clear pulse, */
                /* Start retrace pulse  */
    _enable();                                  /* Interrupts on now    */
    while (inp(MID_STAT) & MIDS_FLYBK);         /* Wait for turnaround  */
    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ));   /* End retrace pulse    */

    /* This is the place to put co-add logic and possibly start the      */
    /* DMA controller for a new scan. Note that the instrument will      */
    /* scan anyway--the decision is whether or not to collect the data. */

    /* Note: May need to insert delay, 10-20ms, to allow for             */
    /* hardware bug in Midac interface, if another scan is to be         */
    /* started here.                                                     */

    outp(PIC1_CMD, PICC_EOI);              /* Issue EOI to master         */
    IoDelay();                             /* Wait a few CPU clocks       */
    if (MidGbl.IrqNum > 7)                 /* If interrupt is on slave    */
        outp(PIC2_CMD, PICC_EOI);            /*   then issue secondary EOI   */

} /* MidAqDmaDone */


/* ---------------------------------------------------------------------- */
/*      MidAqSetGain:   Set Signal Gain                                   */
/*                                                                        */
/* ---------------------------------------------------------------------- */

int MidAqSetGain(int SignalGain)
{
    int gainport = ((~SignalGain << MIDC_GSHIFT) & MIDC_GMASK);
    int oldgain = MidGbl.GainVal;
```

```
    if (SignalGain<0 || SignalGain>7)
            return -1;

    CmdOut(gainport | (CmdIn() & ~MIDC_GMASK));
    MidGbl.GainVal = SignalGain;
    MidGbl.GainPort = gainport;
    return oldgain;

} /* MidAqSetGain */


/* -------------------------------------------------------------------- */
/*                                                                      */
/*      MidAqTerm:      Data collect termination                        */
/*                                                                      */
/*      This function is not explicitly called, but is called at        */
/*      program termination via the atexit() facility. The primary      */
/*      task is to disable DMA and the terminal count interrupt and     */
/*      restore the IRQ vector.                                         */
/* -------------------------------------------------------------------- */

void MidAqTerm(void)
{

    SetIrqEnable(MidGbl.IrqNum, 0);     /* Disable interrupt channel    */
    DisableDma(MidGbl.DmaChannel);      /* Disable DMA channel          */
    CmdOut(MIDC_EOS);                   /* Reset the interferometer     */
    IoDelay();                          /* Wait a few CPU clocks        */

    if (MidGbl.OldIrqVec != NULL)
    {
            _dos_setvect(MidGbl.IrqVecNo, MidGbl.OldIrqVec);
            MidGbl.OldIrqVec = NULL;
    }

} /* MidAqTerm */


/* -------------------------------------------------------------------- */
/*      MidAqInit:      Initialize Midac interface for data collect     */
/*                                                                      */
/*      The arguments to this function provide for setup parameters     */
/*      and/or nonstandard interface board configurations. Each is      */
/*      either a nonnegative integer value, or -1 to use the            */
/*      predefined default value.                                       */
/*                                                                      */
/*      The first two arguments (DmaChannel, IrqNumber) describe the    */
/*      configuration of the Midac interface board. Current interface   */
/*      boards are hardwired for DMA channel 1 and are jumper           */
/*      selectable to use either IRQ2 or IRQ3. Other options could      */
/*      conceivably be possible for unusual custom requirements.        */
/*      In general, however, such a modified interface board would      */
```

```
/*      be incompatible with existing SpectraCalc and LabCalc drivers.  */
/*                                                                      */
/*      The buffer size argument (MaxPoints) is necessary to allocate   */
/*      a DMA buffer. This buffer has the hardware-enforced             */
/*      requirement to not cross a 64K-byte absolute memory boundary.   */
/*      This is the strictest dynamic allocation requirement in a       */
/*      typical data collect application, and should be done first.     */
/*      If co-addition of interferograms is to be performed, this is    */
/*      might be a good place to allocate an accumulator buffer as      */
/*      well.                                                           */
/*                                                                      */
/*      The gain argument (SignalGain) provides the initial signal      */
/*      gain level for programming the interface. This value is         */
/*      subject to change during program operation, but some initial    */
/*      value is required.                                              */
/* -------------------------------------------------------------------- */

int MidAqInit(
    int         DmaChannel,     /* DMA channel number, 0-3              */
    int         IrqNumber,      /* PC/ISA interrupt channel number      */
    int         SignalGain,     /* Signal gain level, 0-7               */
    int         MaxPoints)      /* Max data points in collect buffer    */
{
    int         i, dmachan, irqnum, maxpts, gainval, gainport;

/* Translate and validate input paramters...                            */

    dmachan     = DmaChannel>=0 ? DmaChannel : DMA;
    irqnum      = IrqNumber >=0 ? IrqNumber  : IRQ;
    gainval     = SignalGain>=0 ? SignalGain : GAIN;
    maxpts      = MaxPoints>=0  ? MaxPoints  : BUFPTS;

    if (dmachan != DMA) return -1;      /* ***temp*** need to know page */
                /* register addresses for other */
                /* DMA channels to generalize   */
                /* this for other byte channels */

    if (dmachan<0 || dmachan>3)
            return -1;
    if (irqnum<0 || irqnum>15)
            return -1;
    if (gainval<0 || gainval>7)
            return -1;
    if (maxpts<1 || maxpts>(MAXDMA / 2))
            return -1;

/* Bring the hardware interface to idle state:                          */

    gainport = (~gainval << MIDC_GSHIFT) & MIDC_GMASK;
                /* Compute inverted gain val    */
    MidGbl.GainVal      = gainval;      /* Save requested gain          */
    MidGbl.GainPort     = gainport;     /* Save port image              */
```

```
    CmdOut(gainport | MIDC_EOS);        /* Set gain, DMA off, and        */
                /*   EOS,IRQ strobes off.        */

    SetIrqEnable(irqnum, 0);            /* Disable interrupt channel    */
    DisableDma(dmachan);                /* Disable DMA channel          */
    IoDelay();                          /* Wait a few CPU clocks        */

/* Initialize DMA:                                                      */

    MidGbl.DmaDone       = 0;
    MidGbl.DmaActive     = 0;
    MidGbl.MaxPoints     = maxpts;
    MidGbl.DmaChannel    = dmachan;
    MidGbl.DmaPageReg    = DmaPageTable[dmachan];
    MidGbl.DmaSize       = (long)maxpts * sizeof(unsigned short);
    MidGbl.DmaBuffer     = GetDmaBuffer(MidGbl.DmaSize);
    if (MidGbl.DmaBuffer == NULL)
            return -1;

    for (i=0; i<maxpts; ++i)            /* Put recognizable null data   */
            MidGbl.DmaBuffer[i] = 0xEEEE;   /*   in buffer for debug        */

/* Initialize IRQ channel                                               */

    MidGbl.IrqNum        = irqnum;
    MidGbl.IrqVecNo      = (irqnum<8 ? 0x08 : 0x68) + irqnum;
    MidGbl.OldIrqVec     = _dos_getvect(MidGbl.IrqVecNo);
    _dos_setvect(MidGbl.IrqVecNo, MidAqDmaDone);

    atexit(MidAqTerm);

    return 0;

} /* MidAqInit */
```

```
/* MENU - Module of functions to put menus on the screen and handle keyboard
 * input. To use it, include the MENU.H file in your program. The following
 * functions are public:
 *
 *   Menu       -   Puts a menu on screen and reads input for it
 *   Box        -   Puts a box on screen (fill it yourself)
 *   GetKey     -   Gets ASCII or function key
 *   _outchar   -   Displays character using current text position and color
 *
 * The following structures are defined:
 *
 *   MENU       -   Defines menu colors, box type, and centering
 *   ITEM       -   Defines text of menu item and index of highlight character
 *
 * The global variable "mnuAtrib" has type MENU. Change this variable to
 * change menu appearance.
 */

#include <string.h>
#include <stddef.h>
#include <ctype.h>
#include <graph.h>
#include <bios.h>
#include "menu.h"

/* Prototype for internal function */
static void Itemize( int row, int col, int fCur, ITEM itm, int cBlank );

/* Default menu attribute. The default works for color or B&W. You can
 * override the default value by defining your own MENU variable and
 * assigning it to mnuAtrib, or you can modify specific fields at
 * run time. For example, you could use a different attribute for color
 * than for black and white.
 */
MENU mnuAtrib =
{
    _TBLACK, _TBLACK, _TWHITE, _TBRIGHTWHITE, _TBRIGHTWHITE,
    _TWHITE, _TWHITE, _TBLACK, _TWHITE, _TBLACK,
    TRUE,
    '┌', '┐', '┘', '└', '│', '─'
};

/* Menu - Puts menu on screen and reads menu input from keyboard. When a
 * highlighted hot key or ENTER is pressed, returns the index of the
 * selected menu item.
 *
 * Params: row and col - If "fCentered" attribute of "mnuAtrib" is true,
 *           center row and column of menu; otherwise top left of menu
 *         aItem - array of structure containing the text of each item
 *           and the index of the highlighted hot key
 *         iCur - index of the current selection--pass 0 for first item,
 *           or maintain a static value
```

```
 *
 * Return: The index of the selected item
 *
 * Uses:   mnuAtrib
 */
int Menu( int row, int col, ITEM aItem[], int iCur )
{
    int cItem, cchItem = 2; /* Counts of items and chars per item      */
    int i, iPrev;           /* Indexes - temporary and previous        */
    int acchItem[MAXITEM];  /* Array of counts of character in items   */
    char *pchT;             /* Temporary character pointer            */
    char achHilite[36];     /* Array for highlight characters          */
    unsigned uKey;          /* Unsigned key code                       */
    long bgColor;           /* Screen color, position, and cursor      */
    short fgColor;
    struct rccoord rc;
    unsigned fCursor;

    /* Save screen information. */
    fCursor = _displaycursor( _GCURSOROFF );
    bgColor = _getbkcolor();
    fgColor = _gettextcolor();
    rc = _gettextposition();

    /* Count items, find longest, and put count of each in array. Also,
     * put the highlighted character from each in a string.
     */
    for( cItem = 0; aItem[cItem].achItem[0]; cItem++ )
    {
        acchItem[cItem] = strlen( aItem[cItem].achItem );
        cchItem = (acchItem[cItem] > cchItem) ? acchItem[cItem] : cchItem;
        i = aItem[cItem].iHilite;
        achHilite[cItem] = aItem[cItem].achItem[i];
    }
    cchItem += 2;
    achHilite[cItem] = 0;          /* Null-terminate and lowercase string  */
    strlwr( achHilite );

    /* Adjust if centered, and draw menu box. */
    if( mnuAtrib.fCentered )
    {
        row -= cItem / 2;
        col -= cchItem / 2;
    }
    Box( row++, col++, cItem, cchItem );

    /* Put items on menu. */
    for( i = 0; i < cItem; i++ )
    {
        if( i == iCur )
            Itemize( row + i, col, TRUE, aItem[i], cchItem - acchItem[i] );
        else
```

```
            Itemize( row + i, col, FALSE, aItem[i], cchItem - acchItem[i] );
    }

    while( TRUE )
    {
        /* Wait until a uKey is pressed, then evaluate it. */
        uKey = GetKey( WAIT );
        switch( uKey )
        {
            case U_UP:                      /* Up key       */
                iPrev = iCur;
                iCur = (iCur > 0) ? (--iCur % cItem) : cItem - 1;
                break;
            case U_DN:                      /* Down key     */
                iPrev = iCur;
                iCur = (iCur < cItem) ? (++iCur % cItem) : 0;
                break;
            default:
                if( uKey > 256 )            /* Ignore unknown function key  */
                    continue;
                pchT = strchr( achHilite, (char)tolower( uKey ) );
                if( pchT != NULL )          /* If in highlight string,      */
                    iCur = pchT - achHilite;/*   evaluate and fall through  */
                else
                    continue;               /* Ignore unknown ASCII key     */
            case ENTER:
                _setbkcolor( bgColor );
                _settextcolor( fgColor );
                _settextposition( rc.row, rc.col );
                _displaycursor( fCursor );
                return iCur;
        }
        /* Redisplay current and previous. */
        Itemize( row + iCur, col,
                 TRUE, aItem[iCur], cchItem - acchItem[iCur] );
        Itemize( row + iPrev, col,
                 FALSE, aItem[iPrev], cchItem - acchItem[iPrev] );
    }
}

/* Box - Draw menu box, filling interior with blanks of the border color.
 *
 * Params: row and col - upper left of box
 *         rowLast and colLast - height and width
 *
 * Return: None
 *
 * Uses:   mnuAtrib
 */
void Box( int row, int col, int rowLast, int colLast )
{
    int i;
```

```
    char achT[MAXITEM + 2];         /* Temporary array of characters */

    /* Set color and position. */
    _settextposition( row, col );
    _settextcolor( mnuAtrib.fgBorder );
    _setbkcolor( mnuAtrib.bgBorder );

    /* Draw box top. */
    achT[0] = mnuAtrib.chNW;
    memset( achT + 1, mnuAtrib.chEW, colLast );
    achT[colLast + 1] = mnuAtrib.chNE;
    achT[colLast + 2] = 0;
    _outtext( achT );

    /* Draw box sides and center. */
    achT[0] = mnuAtrib.chNS;
    memset( achT + 1, ' ', colLast );
    achT[colLast + 1] = mnuAtrib.chNS;
    achT[colLast + 2] = 0;
    for( i = 1; i <= rowLast; ++i )
    {
        _settextposition( row + i, col );
        _outtext( achT );
    }

    /* Draw box bottom. */
    _settextposition( row + rowLast + 1, col );
    achT[0] = mnuAtrib.chSW;
    memset( achT + 1, mnuAtrib.chEW, colLast );
    achT[colLast + 1] = mnuAtrib.chSE;
    achT[colLast + 2] = 0;
    _outtext( achT );
}

/* Itemize - Display one selection (item) of a menu. This function
 * is normally only used internally by Menu.
 *
 * Params: row and col - top left of menu
 *         fCur - flag set if item is current selection
 *         itm - structure containing item text and index of highlight
 *         cBlank - count of blanks to fill
 *
 * Return: none
 *
 * Uses:   mnuAtrib
 */
void Itemize( int row, int col, int fCur, ITEM itm, int cBlank )
{
    int i;
    char achT[MAXITEM];             /* Temporary array of characters */

    /* Set text position and color. */
```

```
    _settextposition( row, col );
    if( fCur )
    {
        _settextcolor( mnuAtrib.fgSelect );
        _setbkcolor( mnuAtrib.bgSelect );
    }
    else
    {
        _settextcolor( mnuAtrib.fgNormal );
        _setbkcolor( mnuAtrib.bgNormal );
    }

    /* Display item and fill blanks. */
    strcat( strcpy( achT, " " ), itm.achItem );
    _outtext( achT );
    memset( achT, ' ', cBlank-- );
    achT[cBlank] = 0;
    _outtext( achT );

    /* Set position and color of highlight character, then display it. */
    i = itm.iHilite;
    _settextposition( row, col + i + 1 );
    if( fCur )
    {
        _settextcolor( mnuAtrib.fgSelHilite );
        _setbkcolor( mnuAtrib.bgSelHilite );
    }
    else
    {
        _settextcolor( mnuAtrib.fgNormHilite );
        _setbkcolor( mnuAtrib.bgNormHilite );
    }
    _outchar( itm.achItem[i] );
}

/* GetKey - Gets a key from the keyboard. This routine distinguishes
 * between ASCII keys and function or control keys with different shift
 * states. It also accepts a flag to return immediately if no key is
 * available.
 *
 * Params: fWait - Code to indicate how to handle keyboard buffer:
 *   NO_WAIT     Return 0 if no key in buffer, else return key
 *   WAIT        Return first key if available, else wait for key
 *   CLEAR_WAIT  Throw away any key in buffer and wait for new key
 *
 * Return: One of the following:
 *
 *   Keytype                                High Byte    Low Byte
 *   -------                                ---------    --------
 *   No key available (only with NO_WAIT)       0            0
 *   ASCII value                                0        ASCII code
 *   Unshifted function or keypad               1        scan code
```

```
 *   Shifted function or keypad                    2        scan code
 *   CTRL function or keypad                       3        scan code
 *   ALT function or keypad                        4        scan code
 *
 * Note:   getkey cannot return codes for keys not recognized by BIOS
 *         int 16, such as the CTRL-UP or the 5 key on the numeric keypad.
 */
unsigned GetKey( int fWait )
{
    unsigned uKey, uShift;

    /* If CLEAR_WAIT, drain the keyboard buffer. */
    if( fWait == CLEAR_WAIT )
        while( _bios_keybrd( _KEYBRD_READY ) )
            _bios_keybrd( _KEYBRD_READ );

    /* If NO_WAIT, return 0 if there is no key ready. */
    if( !fWait && !_bios_keybrd( _KEYBRD_READY ) )
        return FALSE;

    /* Get key code. */
    uKey = _bios_keybrd( _KEYBRD_READ );

    /* If low byte is not zero, it's an ASCII key. Check scan code to see
     * if it's on the numeric keypad. If not, clear high byte and return.
     */
    if( uKey & 0x00ff )
        if( (uKey >> 8) < 69 )
            return( uKey & 0x00ff );

    /* For function keys and numeric keypad, put scan code in low byte
     * and shift state codes in high byte.
     */
    uKey >>= 8;
    uShift = _bios_keybrd( _KEYBRD_SHIFTSTATUS ) & 0x000f;
    switch( uShift )
    {
        case 0:
            return( 0x0100 | uKey );  /* None (1)    */
        case 1:
        case 2:
        case 3:
            return( 0x0200 | uKey );  /* Shift (2)   */
        case 4:
            return( 0x0300 | uKey );  /* Control (3) */
        case 8:
            return( 0x0400 | uKey );  /* Alt (4)     */
    }
}

/* _outchar - Display a character. This is the character equivalent of
 * _outtext. It is affected by _settextposition, _settextcolor, and
```

```
 * _setbkcolor. It should not be used in loops. Build strings and then
 * _outtext to show multiple characters.
 *
 * Params: out - character to be displayed
 *
 * Return: none
 */
void _outchar( char out )
{
    static char achT[2] = " ";       /* Temporary array of characters */

    achT[0] = out;
    _outtext( achT );
}
```

Blank

APPENDIX H

MIDAC REAL-TIME PATTERN RECOGNITION PROGRAM

```
/**********************************************************************/
/*
   program MTRX

   This program is a "C" version of the TESTWV program located on the
   Silicon Graphics computer.  The program will process interferograms
   collected on the Midac unit 120 interferometer and display the
   result of the filtering and pattern recognition.

   author: Bob Kroutil, Mike Housky

   date: October 1992   */

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <graph.h>
#include <math.h>
#include <time.h>
#include "headers.def"
#include "mtrx.def"

#include <stddef.h>/* Standard ANSI headers*/
#include <conio.h>/* MSC-specific headers*/
#include <dos.h>

#include "middef.h"/* Midac-specific headers*/
#include "filter1.inc" /* include the digital filter coefficients */
#include "discrim1.inc" /* include the pattern recognition coefficients */

/* ------------------------------------------------------------------- */
/*              Local definitions:                                     */
/* ------------------------------------------------------------------- */

/* MSC7/MSC6 Portability:                                              */

#ifdef MSC_VER
#if MSC_VER >= 700
#define outp _outp
#define inp  _inp
#endif
#endif

#define TIMEOUT 20.0            /* DMA Completion timeout, in seconds   */

/* Defaults for MidAqInit:                                             */

#define DMA             1       /* Default DMA channel                  */
#define DMAPAGE         0x83    /* DMA page register port for default   */
                /*    channel                            */
#define IRQ             2       /* Default IRQ channel                  */
#define GAIN            0       /* Default signal gain level (0-7)      */
```

```
#define BUFPTS          16384   /* Default DMA buffer size in data     */
                /*   points                              */
#define MAXDMA          0xFF80  /* Maximum DMA buffer size in bytes    */

            /* Note: MAXDMA must be less than the "ideal" limit of  */
            /* 64K for the GetDmaBuffer function to work properly.  */

/*
            System board (PC/AT) I/O definitions:
*/

#define SYS_DMA1        0x00    /* Base of byte DMA controller         */

/* These ports are channel-independent:                                */

#define DMA_STAT  (SYS_DMA1+ 8) /* (R) Status register                 */
#define DMA_CMD   (SYS_DMA1+ 8) /* (W) Command register                */
#define DMA_REQ   (SYS_DMA1+ 9) /* (W) Request register                */
#define DMA_WSMR  (SYS_DMA1+10) /* (W) Write single mask register      */
#define DMA_MODE  (SYS_DMA1+11) /* (W) Mode register                   */
#define DMA_CLRF  (SYS_DMA1+12) /* (W) Clear byte pointer flip-flop    */
#define DMA_TEMP  (SYS_DMA1+13) /* (R) Temporary register              */
#define DMA_MCLR  (SYS_DMA1+13) /* (W) Master Clear                    */
#define DMA_CMSK  (SYS_DMA1+14) /* (W) Clear mask register             */
#define DMA_WAMR  (SYS_DMA1+15) /* (W) Write all mask register bits    */

/* These occur 4 times, once for each channel. Add 2*(channel number)  */
/* to get true port address:                                           */

#define DMA_ADDR (SYS_DMA1+ 0)  /* (R/W) Base or current address       */
#define DMA_CTR  (SYS_DMA1+ 1)  /* (R/W) Base or current word count    */

#define SYS_PIC1        0x20    /* Base of primary interrupt controller */
#define PIC1_CMD  (SYS_PIC1+0)  /* (W) Command register (OCW2/OCW3)    */
#define PIC1_STAT (SYS_PIC1+0)  /* (R) Status register (ISR or IRR)    */
#define PIC1_MASK (SYS_PIC1+1)  /* (R/W) Interrupt mask register       */

#define SYS_PIC2        0xA0    /* Base of secondary int. controller   */
#define PIC2_CMD  (SYS_PIC2+0)  /* (W) Command register (OCW2/OCW3)    */
#define PIC2_STAT (SYS_PIC2+0)  /* (R) Status register (ISR or IRR)    */
#define PIC2_MASK (SYS_PIC2+1)  /* (R/W) Interrupt mask register       */

#define PICC_EOI        0x20    /* OCW2 (nonspecific) End-Of-Interrupt */
                /*   command                             */

/*
            Local Macros:
*/

#define PtrToLong(p) (((long)FP_SEG(p) << 4) + (long)FP_OFF(p))
                /* Macro to convert far pointer to       */
                /*   20-bit absolute address             */
```

```
#define DisableDma(ch) outp(DMA_WSMR, (ch)+4)    /* Disable DMA channel  */
#define EnableDma(ch)  outp(DMA_WSMR, (ch))      /* Enable DMA channel   */

/* Input and output from read-only command port, a shadow copy of the   */
/* port value is kept in MidGbl.CmpPort:                                */

#define CmdIn()    (MidGbl.CmdPort)
#define CmdOut(val) (outp(MID_CMD, MidGbl.CmdPort = (int)(val)), \
                     outp(MID_CMD, MidGbl.CmdPort))

/* ------------------------------------------------------------------ */
/*              Global variables:                                     */
/* ------------------------------------------------------------------ */

MidAqGlobalType near MidGbl;    /* Global paramater/context variables   */

static int near DmaPageTable[8] = /* Table of DMA page register ports   */
            { 0x87, 0x83, 0x81, 0x82, -1, 0x8B, 0x89, 0x8A };

#define INTF_LENGTH 1024        /* Length of the interferogram */
#define PLIMIT 1024             /* number of points collected from Midac */
#define GH_LENGTH 512           /* Bytes in the global header */
#define SH_LENGTH 64            /* Bytes in the subfile header */
#define FEND 79                 /* set the key to terminate program */
#define DELAY_LENGTH 256

main(argc,argv)
int argc;
char *argv[];

{
  int MidAqInit(), MidAqSetGain();
  void MidAqStartScan();
  float raw_buf[INTF_LENGTH], intf_buf[INTF_LENGTH], flt_buf[SEG_LENGTH1];
  float plinear(), kalman();
  float delay[DELAY_LENGTH], dsc_result, kal_result=0.0;
  int fburst();
  int scan=-1, index, burst, i, loop=0, igain=-1;
  char ch;
  void deriv(), rotate(), normal(), filter(), lets_see_it();
  void logoega(), grf_results();
  FILE *device, *fp2;
  struct global_header gh;
  struct scan_header sh;
  long time_stamp();
  unsigned long ta0,ta1;
/*  long tm0,tm1,tm2,tm3,tm4,tm5,tm6,tm7,tm8; */
/*  int t0=0,t1=0,t2=0,t3=0,t4=0,t5=0,t6=0,t7=0,t8=0;   */

  if (argc != 2)
    {
    printf("\nUsage: mtrx outfile\n");
```

```
    exit(1);
    }

  /* identify the output device */
  device = stdout;
/* device = stdprn; */

  /* Open a file connection to the results */
  if ((fp2 = fopen(argv[1], "w")) == NULL)
    {
    printf("Unable to open \"%s\"\n", argv[1]);
    exit(1);
    }

  /* Zero-fill the delay line */
  for (i=0; i<DELAY_LENGTH; i++)
    delay[i] = 0.0;

  /* Set up the screen */
  _setvideomode(_ERESCOLOR);
  _setbkcolor(_BLUE);

  /* initialize the Midac interferometer */

    i = MidAqInit( -1, -1, igain, PLIMIT);
    if (i)
    {
            printf("Error: MidAqInit returned %d\n", i);
            return 1;
    }
/*    printf("MidCol initialized:\n");
    printf("  DMA Buffer at %Fp = %06lX\n", MidGbl.DmaBuffer,
                PtrToLong(MidGbl.DmaBuffer)); */

/******************************************************************/
 /* check the instrument gain -- if too low, then increase gain
                              if too high, then decrease gain */
/*    igain++;
    MidAqStartScan();
    ta0 = (unsigned long)clock();
    while (!MidGbl.DmaDone)
     {
      ta1 = (unsigned long)clock();
      if ((ta1-ta0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
        {
         printf("Error: Timeout on DMA completion\n");
         return 2;
        }
     }
    MidGbl.DmaActive = 0;
    for (index=0; index < PLIMIT; index++)
        raw_buf[index] = (float) MidGbl.DmaBuffer[index];
```

```
    burst = fburst(raw_buf,PLIMIT);
    while(fabs(raw_buf[burst]) <= 16384. && igain <= 7)
     {
      raw_buf[burst] *= 2.;
      igain++;
     }
      MidAqSetGain(igain);
      printf(".... setting the instrument A/D gain to = %d",igain);
      MidAqStartScan();
      ta0 = (unsigned long)clock();
      while (!MidGbl.DmaDone)
        {
          ta1 = (unsigned long)clock();
          if ((ta1-ta0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
            {
              printf("Error: Timeout on DMA completion\n");
              return 2;
             }
        }
       MidGbl.DmaActive = 0; */
/*********************************************************************/

/* set up the main loop to process data */

tloop:
    scan++;

    /* Check for exit key */
    if (kbhit() != 0)
      {
      ch = getch();
      if (ch == FEND)
        {
        fclose(fp2);
        _setvideomode(_DEFAULTMODE);

/*      printf ("\n\n");
        printf("Burst/Flip:      %02d\n", t0/scan);
        printf("Derivative:      %02d\n", t1/scan);
        printf("Burst location: %02d\n", t2/scan);
        printf("Normalization:  %02d\n", t3/scan);
        printf("Filter:          %02d\n", t4/scan);
        printf("Discrimination: %02d\n", t5/scan);
        printf("Kalman:          %02d\n", t6/scan);
        printf("Graphics:        %02d\n", t7/scan);
        printf("                ====\n");
        printf("Total time:     %02d ticks or %d mseconds\n",
                t8/scan, (t8/scan)*55);
*/
        exit(1);
        }
      }
```

```
/* Collect 1 sample interferogram trace: */

    MidAqStartScan();
    ta0 = (unsigned long)clock();
    while ( !MidGbl.DmaDone )
    {
            ta1 = (unsigned long)clock();
            if ((ta1 - ta0) > (unsigned long)(TIMEOUT * CLOCKS_PER_SEC))
            {
                printf("Error: Timeout on DMA completion\n");
                return 2;
            }
            /* maybe do something else while waiting */;
    }
    MidGbl.DmaActive = 0;

    /* convert the integer array to a ungain ranged floating array */
    for (index = 0; index < INTF_LENGTH; index++)
      raw_buf[index] =  (float) MidGbl.DmaBuffer[index];
/*    lets_see_it(device, "RAW", raw_buf, INTF_LENGTH); */


    /* Flip interferogram if burst is negative */
/*    tm0 = time_stamp(); */
    burst = fburst(raw_buf);
    if (raw_buf[burst] < 0.0)
      {
      for (i=0; i<INTF_LENGTH; i++)
        raw_buf[i] *= -1.0;
      }

    /* Calculate the derivative of the interferogram */
/*    tm1 = time_stamp(); */
    deriv(intf_buf, raw_buf);
/*    tm2 = time_stamp(); */
/*    lets_see_it(device, "DRV", intf_buf, INTF_LENGTH);   */

    /* find the burst of the interferogram */
    burst = fburst(intf_buf);
/*    tm3 = time_stamp(); */

    /* normalize the interferogram */
    normal(intf_buf);
/*    tm4 = time_stamp(); */
/*    lets_see_it(device, "NML", intf_buf, INTF_LENGTH);   */

    /* filter the short section */
    filter(intf_buf, flt_buf, burst);
/*    tm5 = time_stamp(); */
/*    lets_see_it(device, "FLT", flt_buf, SEG_LENGTH);   */

    /* piece-wise linear discrimant */
```

```
    dsc_result = plinear(flt_buf);
/*    tm6 = time_stamp(); */

    /* kalman filter */
    kal_result = kalman(scan, dsc_result);
/*    tm7 = time_stamp(); */

    if (scan < DELAY_LENGTH)
      delay[scan] = kal_result;
    else
      {
      for (i=1; i<DELAY_LENGTH; i++)
        delay[i-1] = delay[i];
      delay[DELAY_LENGTH-1] = kal_result;
      }

    loop = loop ^ 1;
    _setactivepage(loop);
    _clearscreen(_GCLEARSCREEN);
    _setvieworg(0,0);
    logoega(2,12);
    _setvieworg(64,175);
    grf_results(scan, kal_result, delay);
    _setvisualpage(loop);
/*    tm8 = time_stamp(); */

    fprintf(fp2,"%04d  %10.5f\n", scan, kal_result);

    /* Update the timing totals */
/*    t0 += ((int) tm1 - tm0); */    /* burst/flip */
/*    t1 += ((int) tm2 - tm1); */    /* derivative */
/*    t2 += ((int) tm3 - tm2); */    /* burst location */
/*    t3 += ((int) tm4 - tm3); */    /* normalization */
/*    t4 += ((int) tm5 - tm4); */    /* filter */
/*    t5 += ((int) tm6 - tm5); */    /* discrimination */
/*    t6 += ((int) tm7 - tm6); */    /* kalman filter */
/*    t7 += ((int) tm8 - tm7); */    /* graphics */
/*    t8 += ((int) tm8 - tm0); */    /* total time */

    goto tloop;

}
/*************************** function logoega ******************************/
/*  logoega is a function used to create the CBDA logo for EGA graphics.
    The funtion requires two parameters, the x and y coordinates for the
    first letter "C". If the logo coordinates are outside the exceptable
    range, no logo will be plotted.

   author: John Ditillo
   modofied by: Bob Kroutil

           logoega is based on the "old" CRDEC routine written by
```

```
          John T. Ditillo

   date: October 1992  */

void logoega(y,x)
int y, x;

{
  int xp, yp;

  if (y<23 & y>1 & x<76 & x>2)
    {

    /* draw the logo */
    _settextposition(y,x);
    _outtext ("C");

    _settextposition(y+1,x-1);
    _outtext ("B D");

    _settextposition(y+2,x);
    _outtext ("A");

    /* Calculate first pixel location */
    yp = y * 14 - 16;
    xp = x * 8 - 5;

    /* first benzene */
    _moveto(xp,yp);
    _lineto(xp-8,yp+3);
    _lineto(xp-8,yp+13);
    _lineto(xp,yp+17);
    _lineto(xp+8,yp+13);
    _lineto(xp+8,yp+3);
    _lineto(xp,yp);

    /* second benzene */
    _moveto(xp-8,yp+13);
    _lineto(xp-16,yp+17);
    _lineto(xp-16,yp+27);
    _lineto(xp-8,yp+31);
    _lineto(xp,yp+27);
    _lineto(xp,yp+17);

    /* third benzene */
    _moveto(xp+8,yp+13);
    _lineto(xp+16,yp+17);
    _lineto(xp+16,yp+27);
    _lineto(xp+8,yp+31);
    _lineto(xp,yp+27);

    /* fourth benzene */
```

```
    _moveto(xp-8,yp+31);
    _lineto(xp-8,yp+42);
    _lineto(xp,yp+45);
    _lineto(xp+8,yp+42);
    _lineto(xp+8,yp+31);

  }

}
/***********************  function grf_results *****************************/
#define INIT_MAX .01

void grf_results (scan,kal,buf)
int scan;
float kal;
float buf[];
{
  char buffer[80];
  int i, x, y, numpts, first_x, last_x, xscale;
  float yscale;
  static float max=INIT_MAX;

  /* set the max value */
  if ((fabs((double)kal)) > max)
    max = (float) (fabs((double)kal));

  if (scan < DELAY_LENGTH)
    {
    numpts = scan;
    first_x = 0;
    last_x = DELAY_LENGTH-1;
    }
  else
    {
    numpts = DELAY_LENGTH;
    first_x = scan - (DELAY_LENGTH-1);
    last_x = scan;
    }

  /* Calculate the scaling factor */
  yscale = 150.0/max;
  xscale = 512/DELAY_LENGTH;

  _moveto (0,0);   /* Print the zero axis */
  _lineto (512,0);
  _moveto (0,150);   /* Print the Y axis */
  _lineto (0,-150);

  for(i = 0; i <= 512; i += 64)  /*Print the X axis tick marks */
    {
    _moveto(i, 5);
    _lineto(i, 0);
```

```
    }

  for(i = 150; i >= -150; i -= 150)   /* Print the Y axis tick marks */
    {
    _moveto(-4, i);
    _lineto(0, i);
    }

  /* label the axis */
  sprintf(buffer,"%04d", first_x);
  _settextposition(24,7);
  _outtext(buffer);

   sprintf(buffer,"%04d", last_x);
  _settextposition(24,70);
  _outtext(buffer);

  sprintf(buffer,"% 5.4f", max);
  _settextposition(3,0);
  _outtext(buffer);

  sprintf(buffer,"% 5.4f", 0.0);
  _settextposition(13,0);
  _outtext(buffer);

  sprintf(buffer,"% 5.4f", -max);
  _settextposition(23,0);
  _outtext(buffer);

  sprintf(buffer,"SCAN: %5d              : % 7.5f", scan, kal);
  for (i=0; i < 10; i++)
    buffer[i+13] = hdmsgl[i];
  _settextposition(1,27);
  _outtext(buffer);

  sprintf(buffer,"End key to exit");
  _settextposition(24,35);
  _outtext(buffer);

  /* plot the data */
  _moveto (0, (int) -(buf[0] * yscale));
  for (i=1; i < numpts; i++)
    {
     x = i * xscale;
     y = (int) -(buf[i] * yscale);
     _lineto (x,y);
    }

}
/*************************** function fburst *************************/
int fburst(buffer)
float buffer[];
```

```
{

/*  int index, bloc;
  double bval;

  bloc = 0;
  bval = (double) buffer[0];

  for (index = 1; index < INTF_LENGTH; index++)
    if (fabs((double) buffer[index]) > bval)
      {
      bval = fabs((double) buffer[index]);
      bloc = index;
      }

  return (bloc);  */

  int i, max_loc, min_loc;
  float max_val=0.0, min_val=0.0;

  for (i=0; i<INTF_LENGTH; i++)
    if (buffer[i] > max_val)
      {
      max_val = buffer[i];
      max_loc = i;
      }
    else if (buffer[i] < min_val)
      {
      min_val = buffer[i];
      min_loc = i;
      }

  if (fabs((double) min_val) > max_val)
    return(min_loc);
  else
    return(max_loc);

}
/**************************** function deriv *******************************/
void deriv(buf1, buf2)
float buf1[], buf2[];
{

  int i2n,in,ib,i2b;
  int index, isrt, ifin, ncent;
  float denom;

  /* use the forward difference for the first two points */
  denom = 2.0;
  i2n = 2;
  in = 1;
  for (index=0; index < 2; index++, i2n++, in++)
```

```
    buf1[index] = (-buf2[i2n] + 4.0*buf2[in] - 3.0*buf2[index])/denom;


  /* use the backward difference for the last two points */
  i2b = INTF_LENGTH - 4;
  ib = INTF_LENGTH - 3;
  isrt = INTF_LENGTH - 2;
  for (index=isrt; index < INTF_LENGTH; index++, i2b++, ib++)
    buf1[index] = (buf2[i2b] - 4.0*buf2[ib] +3.0*buf2[index])/denom;


  /* use the central difference for the middle points */
  ncent = INTF_LENGTH - 5;
  isrt = 2;
  ifin = INTF_LENGTH - 2;
  i2b = 0;
  ib = 1;
  in = 3;
  i2n = 4;
  denom = 12.0;
  for (index=isrt; index < ifin; index++, i2n++, in++, ib++, i2b++)
     buf1[index] = (buf2[i2b] - 8.0*buf2[ib] + 8.0*buf2[in] -
buf2[i2n])/denom;


}
/****************************** function normal ******************************/
void normal(buffer)
float buffer[];
{

  int index;
  float ssq = 0.0;

  for (index=0; index < INTF_LENGTH; index++)
    ssq += buffer[index] * buffer[index];

  if (ssq > 0.0)
    ssq = INTF_LENGTH / sqrt(ssq);
  else
    ssq = 1.0;

  for (index=0; index < INTF_LENGTH; index++)
    buffer[index] *= ssq;

}
/************************* function filter *********************************/
void filter(in_buf, out_buf, burst)
float in_buf[];
float out_buf[];
int burst;
{
```

```
  int i, j, k;

  for (i=0, k=SEG_OFFSET1+burst-1; i<SEG_LENGTH1; i++, k++)
    {
    out_buf[i] = flt_intercepts1[i];
    for (j=0; j < flt_length1[i]; j++)
      out_buf[i] += flt_coefs1[i][j] * in_buf[ k+flt_offsets1[i][j] ];
    }
}
/************************* function plinear ********************************/
float plinear(in_buf)
float in_buf[];
{

  float dsc_max=-100.0;
  float dsc;
  int i, j, k;

  for (i=0; i < DSC_PASS1; i++)
    {
    dsc = dsc_intercepts1[i];
    for (j=0; j < SEG_LENGTH1; j++)
      dsc += in_buf[j] * dsc_coefs1[i][j];

    if (dsc > dsc_max)
      dsc_max = dsc;
    }

  return(dsc_max);
}
/************************** function kalman ********************************/
float kalman(scan_num, in_value)
int scan_num;
float in_value;
{
  static float sum=0.0;
  static float sumsq=0.0;
  static float q=0.0;
  static float clkip=0.0;
  static float sigcl2, skip;
  static float prev_input[2*KAL_WIN+1];
  static float beta[2*KAL_WIN+1];
  int nm, i, j;
  float temp, sbase, skm, kal_gain, kal_result, clcov, clkm;

  nm = 2 * KAL_WIN + 1;

  if (scan_num == 0)
    {
    /* setup info for the kalman */
    temp = 3.0/(float)(KAL_WIN*(KAL_WIN+1)*(2*KAL_WIN+1));
    for (i=-KAL_WIN, j=0; i<KAL_WIN+1; i++, j++)
```

```
      beta[j] = temp * (float) i;
    }

  if (scan_num < KAL_SETUP)
    {
    sum += in_value;
    sumsq += in_value * in_value;
    return(0.0);
    }

  else if (scan_num == KAL_SETUP)
    {
    sbase = (float) KAL_SETUP;
    sigcl2 = (sbase * sumsq - sum * sum)/(sbase*(sbase-1.0));
    skip = sigcl2;
    return(0.0);
    }

  else
    {
    clkm = clkip;
    skm = skip + q;

    /* compute the kalman gain */
    kal_gain = skm / (skm + sigcl2);

    /* update the intensity covariance */
    clcov = (1.0 - kal_gain) * skm;

    /* update the intensity estimate */
    kal_result = clkm + kal_gain * (in_value - clkm);

    /* update array of previous values */
    for (i=0; i<nm-1; i++)
      prev_input[i] = prev_input[i+1];
    prev_input[nm-1] = in_value;

    /* update the Q estimate and compute the moving average */
    for (i=0, sum=0.0; i<nm; i++)
      sum += beta[i] * prev_input[i];

    q = sum * sum;
    clkip = kal_result;
    skip = clcov;
    return(kal_result);
    }
}
/************************* function lets_see_it ****************************/
void lets_see_it(device, label, buffer, length)
FILE *device;
char label[];
float buffer[];
```

```
int length;
{
  int i;

  if (device == stdout)         /* Output to display */
    {
    fprintf(device,"\n\n\n\n");
    for (i=0; i < length; i += 2)
    fprintf(device, "%s# %4d = %12.3f       %s# %4d = %12.3f\n",
            label, i+1, buffer[i], label, i+2, buffer[i+1]);
    }
  else                          /* Output to the printer */
    {
    fprintf(device,"\r\n\r\n\r\n\r\n");
    for (i=0; i < length; i += 4)
      {
      fprintf(device, "%s# %4d = %12.3f       %s# %4d = %12.3f    ",
              label, i+1, buffer[i], label, i+2, buffer[i+1]);
      fprintf(device, "%s# %4d = %12.3f       %s# %4d = %12.3f\r\n",
              label, i+3, buffer[i+2], label, i+4, buffer[i+3]);
      }
    }
}
/************************ function time_stamp *****************************/
long time_stamp()
  {
  union REGS regs;                /*SETUP FOR REGISTER USE*/
  long tc;

  regs.h.ah = 0;                  /*SET ACC FOR TIME TYPE INTERRUPT*/
  int86( 0x1a, &regs, &regs );    /*GENERATE INTERRUPT FOR TIME*/
  tc = (((long) regs.x.cx) << 16) + regs.x.dx;
  return(tc);                     /*RETURN CLOCK TICK*/
  }
/* ------------------------------------------------------------------ */
/*     in:    Allow port input during debug.                           */
/*            This is necessary for CV 4.00--the "I" command (port     */
/*            input is broken. The circumvention is to include a       */
/*            a global function such as in() below, trace at least     */
/*            as far as the main() function, then "?in(port)" or       */
/*            "?in(port),x" to read port contents.                     */
/* ------------------------------------------------------------------ */

int in( unsigned port )
{
    int i;
    i = inp(port);
    return i;

} /* in */
```

```
/* -------------------------------------------------------------------- */
/*      IoDelay:        I/O delay for IBM/AT and clones.                */
/*                                                                      */
/*      This dummy function is used to generate a few clocks of delay   */
/*      between consecutive accesses to certain I/O ports. Basically,   */
/*      the call/return sequence is more than enough. Assembler         */
/*      programs typically use a "JMP SHORT $+2" instruction, but       */
/*      the MSC7 inline assembler doesn't seem to handle the "$"        */
/*      token very well. The delay is necessary on IBM AT machines      */
/*      and true compatibles.                                           */
/*                                                                      */
/*      Needless to say, allowing this function to be inlined would     */
/*      be a bad idea...                                                */
/* -------------------------------------------------------------------- */

static void near IoDelay(void)
{
   ;
} /* IoDelay */

/* -------------------------------------------------------------------- */
/*      GetDmaBuffer:   Allocate a byte-DMA compatible buffer           */
/*                                                                      */
/*      A byte DMA buffer cannot cross a 64K-byte absolute address      */
/*      boundary.                                                       */
/*                                                                      */
/*      Returns pointer to buffer if successful, NULL otherwise.        */
/* -------------------------------------------------------------------- */

void far *GetDmaBuffer(long Size)
{
    #define MaxTries 16         /* Maximum attempts before failure      */

    void        far *failed[MaxTries],
                 far *try,
                 far *retry;
    unsigned    begoff, endoff;
    int         i, nfail=0;

    if (Size>MAXDMA || Size<=0) return NULL;

    for (;;)                            /* Repeat until explicit break: */
    {
            try = malloc((size_t)Size);
            if ( try==NULL ) break;

/* Test for 64K block wraparound:                                       */

            begoff = (FP_SEG(try) << 4) + FP_OFF(try);
            endoff = begoff + (unsigned)Size - 1;
            if (endoff >= begoff) break;    /* Success if all in 1 block    */
```

```
/* Current attempt crosses boundary, retry if failed list not full:       */

            if (nfail == MaxTries)
            {
                free(try);
                try = NULL;
                break;
            }

/* Resize current try to end on 64K absolute boundary and add it to      */
/* the failed list:                                                      */

            retry = realloc(try, 1+~begoff);
            if ( retry != NULL )
                try = retry;
            failed[nfail++] = try;
    }

/* Arrive here via explicit break. Free failed attempt pointers, if      */
/* any and exit. The try variable has been set to a pointer on success   */
/* or to NULL on error.                                                  */

    for( i=0; i<nfail; ++i )
    {
            free( failed[i] );
    }

    return try;

#undef MaxTries                           /* Undefine "local" macros      */

} /* GetDmaBuffer */


/* -------------------------------------------------------------------- */
/*      StartDma:       Start a DMA operation.                          */
/*                                                                      */
/*      This is a cut-down version to do input only, specifically       */
/*      using DMA info in MidGbl structure.                             */
/* -------------------------------------------------------------------- */

void StartDma(void)
{
    long        addr = PtrToLong(MidGbl.DmaBuffer);
    int         size = (int)MidGbl.DmaSize;
    unsigned    ch   = 2*MidGbl.DmaChannel;

    DisableDma(MidGbl.DmaChannel);
    IoDelay();                            /* Wait a few CPU clocks        */
    outp(DMA_MODE, 0x44+MidGbl.DmaChannel);
                /* DMA Mode: single-block,      */
                /*   increment address,         */
```

```
                /*   no autoinitialize,        */
                /*   "write transfer" -> cpu   */
    IoDelay();                          /* Wait a few CPU clocks        */

    outp(DMA_CLRF,0);                   /* Set to receive LSB first     */
    IoDelay();                          /* Wait a few CPU clocks        */

    outp(DMA_CTR+ch, (int)size);        /* Send byte count              */
    IoDelay();                          /* Wait a few CPU clocks        */
    outp(DMA_CTR+ch, (int)size >> 8);
    IoDelay();                          /* Wait a few CPU clocks        */

    outp(DMA_ADDR+ch, (int)addr);       /* Send address                 */
    IoDelay();                          /* Wait a few CPU clocks        */
    outp(DMA_ADDR+ch, (int)addr >> 8);
    IoDelay();                          /* Wait a few CPU clocks        */

    outp(MidGbl.DmaPageReg, (int)(addr>>16));
                /* Set page reg to top 8 bits   */
    IoDelay();                          /* Wait a few CPU clocks        */

    EnableDma(MidGbl.DmaChannel);       /* Finally, enable DMA          */

} /* StartDma */


/* ------------------------------------------------------------------ */
/*      SetIrqEnable:   Set/Reset IRQ enable status for specified      */
/*                      channel.                                       */
/*                                                                     */
/*      Please note that the sense of the "Enable" argument is a C-    */
/*      style boolean. Nonzero, or "true", enables the channel. This   */
/*      is opposite from the 8259 mask register, where a 1 disables    */
/*      the channel and 0 enables.                                     */
/* ------------------------------------------------------------------ */

void SetIrqEnable(
    int         IrqNumber,      /* Interrupt channel, 0-15             */
    int         Enable)         /* New enable status for this channel  */
                /*   0 = disable interrupts          */
                /*   nonzero = enable interrupts     */
{
    unsigned    port;
    int         mask, val;

    if (IrqNumber < 8)
    {
        port = PIC1_MASK;               /* Primary 8259 port            */
        mask = 1 << IrqNumber;
    }
    else
    {
```

```
        port = PIC2_MASK;                /* Secondary 8259 port          */
        mask = 1 << (IrqNumber-8);
    }

    val = inp(port) | mask;             /* Set to mask disable          */
    if (Enable) val -= mask;            /* Set to enable if requested   */
    outp(port, val);                    /* Update port                  */

} /* SetIrqEnable */


/* --------------------------------------------------------------------- */
/*      MidAqStartScan: Start new data collect operation                 */
/*                                                                       */
/*      This is a skeleton of what is needed to begin a new data         */
/*      scan, or series of accumulated scans, on the Midac FT-IR.        */
/* --------------------------------------------------------------------- */

void MidAqStartScan(void)
{
    SetIrqEnable(MidGbl.IrqNum, 0);     /* Disable interrupt channel    */
    IoDelay();                          /* Wait a few CPU clocks        */
    DisableDma(MidGbl.DmaChannel);      /* Disable DMA channel          */
    IoDelay();                          /* Wait a few CPU clocks        */

    StartDma();                         /* Start DMA channel            */

    SetIrqEnable(MidGbl.IrqNum, 1);     /* Enable interrupt channel     */

/* Set gain and retrace interferometer:                                 */

    CmdOut( MidGbl.GainPort | MIDC_EOS | MIDC_IRQ );
                /* Start IRQ clear pulse*/
    IoDelay();                                  /* Wait a few CPU clocks*/
    CmdOut( CmdIn() &~(MIDC_EOS + MIDC_IRQ) );  /* End IRQ clear pulse, */
                /* Start retrace pulse  */
    IoDelay();                                  /* Wait a few CPU clocks*/
    while (inp(MID_STAT) & MIDS_FLYBK);         /* Wait for turnaround  */
    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ));   /* End retrace pulse    */
    IoDelay();                                  /* Wait a few CPU clocks*/

    /* Note: May need to insert delay here, 10-20ms, to allow for       */
    /* hardware bug in Midac interface causing early DMA requests.      */
             _asm xor  cx,cx
        here:   _asm loop here

    MidGbl.DmaActive = 1;               /* Set global DMA status flags  */
    MidGbl.DmaDone = 0;

    CmdOut( CmdIn() | MIDC_DMA );       /* Enable DMA at interface      */

} /* MidAqStartScan */
```

```
/* ----------------------------------------------------------------- */
/*      MidAqDmaDone:   Interrupt Handler for DMA completion        */
/*                                                                   */
/*      This version simply notes DMA completion, retraces the      */
/*      interferometer, and disables DMA at both the 8237 and at    */
/*      the Midac interface board.  This would be the natural place  */
/*      to insert co-add logic for averaging interferograms.        */
/* ----------------------------------------------------------------- */

void _cdecl _interrupt far MidAqDmaDone(void)
{
    MidGbl.DmaDone = 1;                    /* Note DMA completion          */

    CmdOut( CmdIn() &~MIDC_DMA );          /* Disable DMA at interface     */
    DisableDma(MidGbl.DmaChannel);         /*  then disable channel        */
    IoDelay();                             /* Wait a few CPU clocks        */

/* Retrace interferometer:                                                 */

    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ) );  /* Start IRQ clear pulse*/
    CmdOut( CmdIn() &~(MIDC_EOS + MIDC_IRQ) );  /* End IRQ clear pulse, */
                /* Start retrace pulse  */
    _enable();                                  /* Interrupts on now    */
    while (inp(MID_STAT) & MIDS_FLYBK);         /* Wait for turnaround  */
    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ));   /* End retrace pulse    */

    /* This is the place to put co-add logic and possibly start the     */
    /* DMA controller for a new scan. Note that the instrument will     */
    /* scan anyway--the decision is whether or not to collect the data. */

    /* Note: May need to insert delay, 10-20ms, to allow for            */
    /* hardware bug in Midac interface, if another scan is to be        */
    /* started here.                                                    */

    outp(PIC1_CMD, PICC_EOI);              /* Issue EOI to master          */
    IoDelay();                             /* Wait a few CPU clocks        */
    if (MidGbl.IrqNum > 7)                 /* If interrupt is on slave     */
            outp(PIC2_CMD, PICC_EOI);          /*   then issue secondary EOI   */

} /* MidAqDmaDone */


/* ----------------------------------------------------------------- */
/*      MidAqSetGain:   Set Signal Gain                              */
/*                                                                   */
/* ----------------------------------------------------------------- */

int MidAqSetGain(int SignalGain)
{
    int gainport = ((~SignalGain << MIDC_GSHIFT) & MIDC_GMASK);
    int oldgain = MidGbl.GainVal;
```

```
    if (SignalGain<0 || SignalGain>7)
            return -1;

    CmdOut(gainport | (CmdIn() & ~MIDC_GMASK));
    MidGbl.GainVal = SignalGain;
    MidGbl.GainPort = gainport;
    return oldgain;

} /* MidAqSetGain */


/* --------------------------------------------------------------------- */
/*                                                                       */
/*      MidAqTerm:      Data collect termination                         */
/*                                                                       */
/*      This function is not explicitly called, but is called at         */
/*      program termination via the atexit() facility. The primary       */
/*      task is to disable DMA and the terminal count interrupt and      */
/*      restore the IRQ vector.                                          */
/* --------------------------------------------------------------------- */

void MidAqTerm(void)
{

    SetIrqEnable(MidGbl.IrqNum, 0);      /* Disable interrupt channel     */
    DisableDma(MidGbl.DmaChannel);       /* Disable DMA channel           */
    CmdOut(MIDC_EOS);                    /* Reset the interferometer      */
    IoDelay();                           /* Wait a few CPU clocks         */

    if (MidGbl.OldIrqVec != NULL)
    {
            _dos_setvect(MidGbl.IrqVecNo, MidGbl.OldIrqVec);
            MidGbl.OldIrqVec = NULL;
    }

} /* MidAqTerm */


/* --------------------------------------------------------------------- */
/*      MidAqInit:      Initialize Midac interface for data collect      */
/*                                                                       */
/*      The arguments to this function provide for setup parameters     */
/*      and/or nonstandard interface board configurations. Each is       */
/*      either a nonnegative integer value, or -1 to use the             */
/*      predefined default value.                                        */
/*                                                                       */
/*      The first two arguments (DmaChannel, IrqNumber) describe the     */
/*      configuration of the Midac interface board. Current interface    */
/*      boards are hardwired for DMA channel 1 and are jumper            */
/*      selectable to use either IRQ2 or IRQ3. Other options could       */
/*      conceivably be possible for unusual custom requirements.         */
/*      In general, however, such a modified interface board would       */
```

```
/*      be incompatible with existing SpectraCalc and LabCalc drivers.  */
/*                                                                      */
/*      The buffer size argument (MaxPoints) is necessary to allocate   */
/*      a DMA buffer. This buffer has the hardware-enforced             */
/*      requirement to not cross a 64K-byte absolute memory boundary.   */
/*      This is the strictest dynamic allocation requirement in a       */
/*      typical data collect application, and should be done first.     */
/*      If co-addition of interferograms is to be performed, this is    */
/*      might be a good place to allocate an accumulator buffer as      */
/*      well.                                                           */
/*                                                                      */
/*      The gain argument (SignalGain) provides the initial signal      */
/*      gain level for programming the interface. This value is         */
/*      subject to change during program operation, but some initial    */
/*      value is required.                                              */
/* -------------------------------------------------------------------- */

int MidAqInit(
    int         DmaChannel,     /* DMA channel number, 0-3              */
    int         IrqNumber,      /* PC/ISA interrupt channel number      */
    int         SignalGain,     /* Signal gain level, 0-7               */
    int         MaxPoints)      /* Max data points in collect buffer    */
{
    int         i, dmachan, irqnum, maxpts, gainval, gainport;

/* Translate and validate input paramters...                            */

    dmachan     = DmaChannel>=0 ? DmaChannel : DMA;
    irqnum      = IrqNumber >=0 ? IrqNumber  : IRQ;
    gainval     = SignalGain>=0 ? SignalGain : GAIN;
    maxpts      = MaxPoints>=0  ? MaxPoints  : BUFPTS;

    if (dmachan != DMA) return -1;      /* ***temp*** need to know page */
                /* register addresses for other */
                /* DMA channels to generalize   */
                /* this for other byte channels */

    if (dmachan<0 || dmachan>3)
            return -1;
    if (irqnum<0 || irqnum>15)
            return -1;
    if (gainval<0 || gainval>7)
            return -1;
    if (maxpts<1 || maxpts>(MAXDMA / 2))
            return -1;

/* Bring the hardware interface to idle state:                          */

    gainport = (~gainval << MIDC_GSHIFT) & MIDC_GMASK;
                /* Compute inverted gain val    */
    MidGbl.GainVal      = gainval;      /* Save requested gain          */
    MidGbl.GainPort     = gainport;     /* Save port image              */
```

```
    CmdOut(gainport | MIDC_EOS);        /* Set gain, DMA off, and       */
                /*   EOS,IRQ strobes off.       */

    SetIrqEnable(irqnum, 0);            /* Disable interrupt channel    */
    DisableDma(dmachan);                /* Disable DMA channel          */
    IoDelay();                          /* Wait a few CPU clocks        */

/* Initialize DMA:                                                      */

    MidGbl.DmaDone       = 0;
    MidGbl.DmaActive     = 0;
    MidGbl.MaxPoints     = maxpts;
    MidGbl.DmaChannel    = dmachan;
    MidGbl.DmaPageReg    = DmaPageTable[dmachan];
    MidGbl.DmaSize       = (long)maxpts * sizeof(unsigned short);
    MidGbl.DmaBuffer     = GetDmaBuffer(MidGbl.DmaSize);
    if (MidGbl.DmaBuffer == NULL)
            return -1;

    for (i=0; i<maxpts; ++i)            /* Put recognizable null data   */
            MidGbl.DmaBuffer[i] = 0xEEEE;   /*   in buffer for debug        */

/* Initialize IRQ channel                                               */

    MidGbl.IrqNum        = irqnum;
    MidGbl.IrqVecNo      = (irqnum<8 ? 0x08 : 0x68) + irqnum;
    MidGbl.OldIrqVec     = _dos_getvect(MidGbl.IrqVecNo);
    _dos_setvect(MidGbl.IrqVecNo, MidAqDmaDone);

    atexit(MidAqTerm);

    return 0;

} /* MidAqInit */
```

```
/***********************************************************************/
/*
    Program MTRXD

    This program is a "C" version of the TESTWV program located
    on the Silicon Graphics computer.  This program will process
    interferograms collected on disk and display the result
    of the filtering and pattern recognition.

    author of modified C version:  Bob Kroutil

    date: October 1992   */

/***********************************************************************/

#include <stdio.h>
#include <fcntl.h>
#include <math.h>
#include <bios.h>
#include <graph.h>
#include <dos.h>
#include "headers.def"
#include "mtrx.def"
#include "filter1.inc"  /* include the filter coefficients */
#include "discrim1.inc" /* include the pattern recognition coefficients */

#define INTF_LENGTH 1024       /* Length of the interferogram */
#define GH_LENGTH 512          /* Bytes in the global header */
#define SH_LENGTH 64           /* Bytes in the subfile header */
#define FEND 79
#define DELAY_LENGTH 256

main(argc,argv)
int argc;
char *argv[];

{
  float raw_buf[INTF_LENGTH], intf_buf[INTF_LENGTH], flt_buf[SEG_LENGTH1];
  float plinear(), kalman();
  float delay[DELAY_LENGTH], dsc_result, kal_result=0.0;
  int fburst();
  int raw_data[INTF_LENGTH];
  int scan, index, burst, i, loop=0;
  int fp1;
  char ch;
  void deriv(), rotate(), normal(), filter(), lets_see_it();
  void logoega(), grf_results();
  FILE *device, *fp2;
  struct global_header gh;
  struct scan_header sh;
  long time_stamp();
  long tm0,tm1,tm2,tm3,tm4,tm5,tm6,tm7,tm8;
```

```
int t0=0,t1=0,t2=0,t3=0,t4=0,t5=0,t6=0,t7=0,t8=0;
union REGS inregs;   /* REG structure for timing input */
union REGS outregs;  /* REG structure for timing output */

if (argc != 3)
  {
  printf("\nUsage: mtrxd infile outfile\n");
  exit(1);
  }

/* prompt user for the output device */
/*  printf("Enter the desired output device for intermediate results\n");
printf("(S)creen or (P)rinter >> ");
ch = getchar();
while (getchar() !='\n');
if (ch == 'P' | ch =='p')
  device = stdprn;
else
  device = stdout; */
device = stdout;


/* Open a file connection to the Midac data file */
if ((fp1 = open(argv[1], O_RDONLY|O_BINARY)) < 0)
  {
  printf("\n\"mtrxd\" is unable to open %s\n",argv[1]);
  exit(1);
  }

/* Open a file connection to the results */
if ((fp2 = fopen(argv[2], "w")) == NULL)
  {
  printf("Unable to open \"%s\"\n", argv[2]);
  exit(1);
  }
else
  fprintf(fp2,"%s\n", argv[1]);

/* Zero-fill the delay line */
for (i=0; i<DELAY_LENGTH; i++)
  delay[i] = 0.0;

/* Set up the screen */
_setvideomode(_ERESCOLOR);
_setbkcolor(_BLUE);

/* read in the global header */
read(fp1, &gh, GH_LENGTH);

for (scan = 0; scan < gh.stop_scan; scan++)
  {
```

```
    /* if using a 486 computer then delay each calculation for
       display purposes -- remove this section for 386 version */
     inregs.h.ah = 0x86;     /* delay service */
     inregs.x.cx = 5;        /* set high order delay word */
     inregs.x.dx = 0;        /* set low order delay word */
     int86 (0x15,&inregs,&outregs); /* call to ROM BIOS timer delay service */

    /* Check for exit key */
    if (kbhit() != 0)
      {
      ch = getch();
      if (ch == FEND)
        {
        fclose(fp2);
        close(fp1);
        _setvideomode(_DEFAULTMODE);
        exit(1);
        }
      }

    read(fp1, &sh, SH_LENGTH);                /* read the subfile header */

    read(fp1, raw_data, INTF_LENGTH*2);       /* read the subfile data */

    /* convert the integer array to a ungain ranged floating array */
    for (index = 0; index < INTF_LENGTH; index++)
      raw_buf[index] =  (float) raw_data[index];
/*    lets_see_it(device, "RAW", raw_buf, INTF_LENGTH); */


    /* Flip interferogram if burst is negative */
    tm0 = time_stamp();
    burst = fburst(raw_buf);
    if (raw_buf[burst] < 0.0)
      for (i=0; i<INTF_LENGTH; i++)
        raw_buf[i] *= -1.0;

    /* Calculate the derivative of the interferogram */
    tm1 = time_stamp();
    deriv(intf_buf, raw_buf);
    tm2 = time_stamp();
/*    lets_see_it(device, "DRV", intf_buf, INTF_LENGTH); */

    /* find the burst of the interferogram */
    burst = fburst(intf_buf);
    tm3 = time_stamp();

    /* normalize the interferogram */
    normal(intf_buf);
    tm4 = time_stamp();
/*    lets_see_it(device, "NML", intf_buf, INTF_LENGTH); */
```

```
    /* filter the short section */
    filter(intf_buf, flt_buf, burst);
    tm5 = time_stamp();
/*    lets_see_it(device, "FLT", flt_buf, SEG_LENGTH); */

    /* piece-wise linear discrimant */
    dsc_result = plinear(flt_buf);
    tm6 = time_stamp();

    /* kalman filter */
    kal_result = kalman(scan, dsc_result);
    tm7 = time_stamp();

    if (scan < DELAY_LENGTH)
      delay[scan] = kal_result;
    else
      {
      for (i=1; i<DELAY_LENGTH; i++)
        delay[i-1] = delay[i];
      delay[DELAY_LENGTH-1] = kal_result;
      }

    loop = loop ^ 1;
    _setactivepage(loop);
    _clearscreen(_GCLEARSCREEN);
    _setvieworg(0,0);
    logoega(2,12);
    _setvieworg(64,175);
    grf_results(scan, kal_result, delay);
    _setvisualpage(loop);
    tm8 = time_stamp();

    fprintf(fp2,"%04d  %10.5f\n", scan, kal_result);

    /* Update the timing totals */
    t0 += ((int) tm1 - tm0);     /* burst/flip */
    t1 += ((int) tm2 - tm1);     /* derivative */
    t2 += ((int) tm3 - tm2);     /* burst location */
    t3 += ((int) tm4 - tm3);     /* normalization */
    t4 += ((int) tm5 - tm4);     /* filter */
    t5 += ((int) tm6 - tm5);     /* discrimination */
    t6 += ((int) tm7 - tm6);     /* kalman filter */
    t7 += ((int) tm8 - tm7);     /* graphics */
    t8 += ((int) tm8 - tm0);     /* total time */
    }
  close(fp1);
  fclose(fp2);
  _setvideomode(_DEFAULTMODE);

  printf("\n\n");
  printf("Burst/Flip:      %02d\n", t0/scan);
  printf("Derivative:      %02d\n", t1/scan);
```

```
  printf("Burst location:  %02d\n", t2/scan);
  printf("Normalization:   %02d\n", t3/scan);
  printf("Filter:          %02d\n", t4/scan);
  printf("Discrimination:  %02d\n", t5/scan);
  printf("Kalman:          %02d\n", t6/scan);
  printf("Graphics:        %02d\n", t7/scan);
  printf("                ====\n");
  printf("Total time:      %02d ticks or  %d mseconds\n",
          t8/scan, (t8/scan)*55);

}
/**************************** function logoega ******************************/
/*  logoega is a function used to create the CBDA logo for EGA graphics.
    The funtion requires two parameters, the x and y coordinates for the
    first letter "C". If the logo coordinates are outside the exceptable
    range, no logo will be plotted.

   author: John Ditillo
   modified by: Bob Kroutil

           logoega is based on the "old" CRDEC logo program
           written by John T. Ditillo

   date: October 1992  */

void logoega(y,x)
int y, x;

{
  int xp, yp;

  if (y<23 & y>1 & x<76 & x>2)
    {

    /* draw the logo */
    _settextposition(y,x);
    _outtext ("C");

    _settextposition(y+1,x-1);
    _outtext ("B D");

    _settextposition(y+2,x);
    _outtext ("A");

    /* Calculate first pixel location */
    yp = y * 14 - 16;
    xp = x * 8 - 5;

    /* first benzene */
    _moveto(xp,yp);
    _lineto(xp-8,yp+3);
    _lineto(xp-8,yp+13);
```

```
    _lineto(xp,yp+17);
    _lineto(xp+8,yp+13);
    _lineto(xp+8,yp+3);
    _lineto(xp,yp);

    /* second benzene */
    _moveto(xp-8,yp+13);
    _lineto(xp-16,yp+17);
    _lineto(xp-16,yp+27);
    _lineto(xp-8,yp+31);
    _lineto(xp,yp+27);
    _lineto(xp,yp+17);

    /* third benzene */
    _moveto(xp+8,yp+13);
    _lineto(xp+16,yp+17);
    _lineto(xp+16,yp+27);
    _lineto(xp+8,yp+31);
    _lineto(xp,yp+27);

    /* fourth benzene */
    _moveto(xp-8,yp+31);
    _lineto(xp-8,yp+42);
    _lineto(xp,yp+45);
    _lineto(xp+8,yp+42);
    _lineto(xp+8,yp+31);

  }

}
/***********************  function grf_results ****************************/
#define INIT_MAX .01

void grf_results (scan,kal,buf)
int scan;
float kal;
float buf[];
{
  char buffer[80];
  int i, x, y, numpts, first_x, last_x, xscale;
  float yscale;
  static float max=INIT_MAX;

  /* set the max value */
  if ((fabs((double)kal)) > max)
    max = (float) (fabs((double)kal));

  if (scan < DELAY_LENGTH)
    {
    numpts = scan;
    first_x = 0;
    last_x = DELAY_LENGTH-1;
```

```
  }
else
  {
  numpts = DELAY_LENGTH;
  first_x = scan - (DELAY_LENGTH-1);
  last_x = scan;
  }

/* Calculate the scaling factor */
yscale = 150.0/max;
xscale = 512/DELAY_LENGTH;

_moveto (0,0);   /* Print the zero axis */
_lineto (512,0);
_moveto (0,150);   /* Print the Y axis */
_lineto (0,-150);

for(i = 0; i <= 512; i += 64)  /*Print the X axis tick marks */
  {
  _moveto(i, 5);
  _lineto(i, 0);
  }

for(i = 150; i >= -150; i -= 150)   /* Print the Y axis tick marks */
  {
  _moveto(-4, i);
  _lineto(0, i);
  }

/* label the axis */
sprintf(buffer,"%04d", first_x);
_settextposition(24,7);
_outtext(buffer);

 sprintf(buffer,"%04d", last_x);
_settextposition(24,70);
_outtext(buffer);

sprintf(buffer,"% 5.4f", max);
_settextposition(3,0);
_outtext(buffer);

sprintf(buffer,"% 5.4f", 0.0);
_settextposition(13,0);
_outtext(buffer);

sprintf(buffer,"% 5.4f", -max);
_settextposition(23,0);
_outtext(buffer);

sprintf(buffer,"SCAN: %5d            : % 7.5f", scan, kal);
for (i =0; i < 10; i++)
```

```
    buffer[i+13]=hdmsg1[i];
  _settextposition(1,27);
  _outtext(buffer);

  sprintf(buffer,"End key to exit");
  _settextposition(24,35);
  _outtext(buffer);

  /* plot the data */
  _moveto (0, (int) -(buf[0] * yscale));
  for (i=1; i < numpts; i++)
    {
     x = i * xscale;
     y = (int) -(buf[i] * yscale);
     _lineto (x,y);
    }

}
/**************************** function fburst **************************/
int fburst(buffer)
float buffer[];
{

/*  int index, bloc;
  double bval;

  bloc = 0;
  bval = (double) buffer[0];

  for (index = 1; index < INTF_LENGTH; index++)
    if (fabs((double) buffer[index]) > bval)
      {
      bval = fabs((double) buffer[index]);
      bloc = index;
      }

  return (bloc);  */

  int i, max_loc, min_loc;
  float max_val=0.0, min_val=0.0;

  for (i=0; i<INTF_LENGTH; i++)
    if (buffer[i] > max_val)
      {
      max_val = buffer[i];
      max_loc = i;
      }
    else if (buffer[i] < min_val)
      {
      min_val = buffer[i];
      min_loc = i;
      }
```

```
  if (fabs((double) min_val) > max_val)
    return(min_loc);
  else
    return(max_loc);

}
/*************************** function deriv ******************************/
void deriv(buf1, buf2)
float buf1[], buf2[];
{

  int i2n,in,ib,i2b;
  int index, isrt, ifin, ncent;
  float denom;

  /* use the forward difference for the first two points */
  denom = 2.0;
  i2n = 2;
  in = 1;
  for (index=0; index < 2; index++, i2n++, in++)
    buf1[index] = (-buf2[i2n] + 4.0*buf2[in] - 3.0*buf2[index])/denom;


  /* use the backward difference for the last two points */
  i2b = INTF_LENGTH - 4;
  ib = INTF_LENGTH - 3;
  isrt = INTF_LENGTH - 2;
  for (index=isrt; index < INTF_LENGTH; index++, i2b++, ib++)
    buf1[index] = (buf2[i2b] - 4.0*buf2[ib] +3.0*buf2[index])/denom;


  /* use the central difference for the middle points */
  ncent = INTF_LENGTH - 5;
  isrt = 2;
  ifin = INTF_LENGTH - 2;
  i2b = 0;
  ib = 1;
  in = 3;
  i2n = 4;
  denom = 12.0;
  for (index=isrt; index < ifin; index++, i2n++, in++, ib++, i2b++)
     buf1[index] = (buf2[i2b] - 8.0*buf2[ib] + 8.0*buf2[in] -
buf2[i2n])/denom;


}
/***************************** function normal ******************************/
void normal(buffer)
float buffer[];
{

  int index;
```

```
  float ssq = 0.0;

  for (index=0; index < INTF_LENGTH; index++)
    ssq += buffer[index] * buffer[index];

  if (ssq > 0.0)
    ssq = INTF_LENGTH / sqrt(ssq);
  else
    ssq = 1.0;

  for (index=0; index < INTF_LENGTH; index++)
    buffer[index] *= ssq;

}
/************************* function filter **********************************/
void filter(in_buf, out_buf, burst)
float in_buf[];
float out_buf[];
int burst;
{

  int i, j, k;

  for (i=0, k=SEG_OFFSET1+burst-1; i<SEG_LENGTH1; i++, k++)
    {
    out_buf[i] = flt_intercepts1[i];
    for (j=0; j < flt_length1[i]; j++)
      out_buf[i] += flt_coefs1[i][j] * in_buf[ k+flt_offsets1[i][j] ];
    }
}
/************************* function plinear *******************************/
float plinear(in_buf)
float in_buf[];
{

  float dsc_max=-100.0;
  float dsc;
  int i, j, k;

  for (i=0; i < DSC_PASS1; i++)
    {
    dsc = dsc_intercepts1[i];
    for (j=0; j < SEG_LENGTH1; j++)
      dsc += in_buf[j] * dsc_coefs1[i][j];

    if (dsc > dsc_max)
      dsc_max = dsc;
    }

  return(dsc_max);
}
/************************* function kalman *******************************/
```

```
float kalman(scan_num, in_value)
int scan_num;
float in_value;
{
  static float sum=0.0;
  static float sumsq=0.0;
  static float q=0.0;
  static float clkip=0.0;
  static float sigcl2, skip;
  static float prev_input[2*KAL_WIN+1];
  static float beta[2*KAL_WIN+1];
  int nm, i, j;
  float temp, sbase, skm, kal_gain, kal_result, clcov, clkm;

  nm = 2 * KAL_WIN + 1;

  if (scan_num == 0)
    {
    /* setup info for the kalman */
    temp = 3.0/(float)(KAL_WIN*(KAL_WIN+1)*(2*KAL_WIN+1));
    for (i=-KAL_WIN, j=0; i<KAL_WIN+1; i++, j++)
      beta[j] = temp * (float) i;
    }

  if (scan_num < KAL_SETUP)
    {
    sum += in_value;
    sumsq += in_value * in_value;
    return(0.0);
    }

  else if (scan_num == KAL_SETUP)
    {
    sbase = (float) KAL_SETUP;
    sigcl2 = (sbase * sumsq - sum * sum)/(sbase*(sbase-1.0));
    skip = sigcl2;
    return(0.0);
    }

  else
    {
    clkm = clkip;
    skm = skip + q;

    /* compute the kalman gain */
    kal_gain = skm / (skm + sigcl2);

    /* update the intensity covariance */
    clcov = (1.0 - kal_gain) * skm;

    /* update the intensity estimate */
    kal_result = clkm + kal_gain * (in_value - clkm);
```

```
    /* update array of previous values */
    for (i=0; i<nm-1; i++)
      prev_input[i] = prev_input[i+1];
    prev_input[nm-1] = in_value;

    /* update the Q estimate and compute the moving average */
    for (i=0, sum=0.0; i<nm; i++)
      sum += beta[i] * prev_input[i];

    q = sum * sum;
    clkip = kal_result;
    skip = clcov;
    return(kal_result);
    }
}
/************************ function lets_see_it **************************/
void lets_see_it(device, label, buffer, length)
FILE *device;
char label[];
float buffer[];
int length;
{
  int i;

  if (device == stdout)        /* Output to display */
    {
    fprintf(device,"\n\n\n\n");
    for (i=0; i < length; i += 2)
    fprintf(device, "%s# %4d = %12.3f      %s# %4d = %12.3f\n",
            label, i+1, buffer[i], label, i+2, buffer[i+1]);
    }
  else                         /* Output to the printer */
    {
    fprintf(device,"\r\n\r\n\r\n\r\n");
    for (i=0; i < length; i += 4)
      {
      fprintf(device, "%s# %4d = %12.3f      %s# %4d = %12.3f    ",
              label, i+1, buffer[i], label, i+2, buffer[i+1]);
      fprintf(device, "%s# %4d = %12.3f      %s# %4d = %12.3f\r\n",
              label, i+3, buffer[i+2], label, i+4, buffer[i+3]);
      }
    }
}
/************************ function time_stamp ****************************/
long time_stamp()
  {
  union REGS regs;             /*SETUP FOR REGISTER USE*/
  long tc;

  regs.h.ah = 0;               /*SET ACC FOR TIME TYPE INTERRUPT*/
  int86( 0x1a, &regs, &regs ); /*GENERATE INTERRUPT FOR TIME*/
  tc = (((long) regs.x.cx) << 16) + regs.x.dx;
```

```
return(tc);                          /*RETURN CLOCK TICK*/
}
```

APPENDIX I

MIDAC DATA COLLECTION AND PATTERN RECOGNITION PROGRAM
FOR IDENTIFYING TWO COMPOUNDS SIMULTANEOUSLY

```
/************************************************************************/
/*
   program MTRX2

   This program is a "C" version of the TESTWV program located on the
   Silicon Graphics computer.  The program will process interferograms
   collected on the Midac unit 120 interferometer and display the
   result of the filtering and pattern recognition.

   author: Bob Kroutil, Mike Housky

   date: March 1993   */

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <graph.h>
#include <math.h>
#include <time.h>
#include "headers.def"
#include "mtrx.def"

#include <stddef.h>/* Standard ANSI headers*/
#include <conio.h>/* MSC-specific headers*/
#include <dos.h>

#include "middef.h"/* Midac-specific headers*/
#include "filter1.inc" /* include digital filter 1 */
#include "discrim1.inc" /* include the pattern recognition coefficients 1 */
#include "filter2.inc" /* include digital filter 2 */
#include "discrim2.inc" /* include the pattern recognition coefficients 2 */

/* --------------------------------------------------------------------- */
/*              Local definitions:                                       */
/* --------------------------------------------------------------------- */

/* MSC7/MSC6 Portability:                                                */

#ifdef MSC_VER
#if MSC_VER >= 700
#define outp _outp
#define inp  _inp
#endif
#endif

#define TIMEOUT 20.0            /* DMA Completion timeout, in seconds    */

/* Defaults for MidAqInit:                                               */

#define DMA             1       /* Default DMA channel                   */
#define DMAPAGE         0x83    /* DMA page register port for default    */
                 /*    channel                       */
```

```
#define IRQ             2       /* Default IRQ channel                        */
#define GAIN            0       /* Default signal gain level (0-7)            */
#define BUFPTS          16384   /* Default DMA buffer size in data            */
                /*   points                                   */
#define MAXDMA          0xFF80  /* Maximum DMA buffer size in bytes           */

            /* Note: MAXDMA must be less than the "ideal" limit of  */
            /* 64K for the GetDmaBuffer function to work properly.  */

/*
            System board (PC/AT) I/O definitions:
*/

#define SYS_DMA1        0x00    /* Base of byte DMA controller                */

/* These ports are channel-independent:                                       */

#define DMA_STAT  (SYS_DMA1+ 8) /* (R) Status register                        */
#define DMA_CMD   (SYS_DMA1+ 8) /* (W) Command register                       */
#define DMA_REQ   (SYS_DMA1+ 9) /* (W) Request register                       */
#define DMA_WSMR  (SYS_DMA1+10) /* (W) Write single mask register             */
#define DMA_MODE  (SYS_DMA1+11) /* (W) Mode register                          */
#define DMA_CLRF  (SYS_DMA1+12) /* (W) Clear byte pointer flip-flop           */
#define DMA_TEMP  (SYS_DMA1+13) /* (R) Temporary register                     */
#define DMA_MCLR  (SYS_DMA1+13) /* (W) Master Clear                           */
#define DMA_CMSK  (SYS_DMA1+14) /* (W) Clear mask register                    */
#define DMA_WAMR  (SYS_DMA1+15) /* (W) Write all mask register bits           */

/* These occur 4 times, once for each channel. Add 2*(channel number)         */
/* to get true port address:                                                  */

#define DMA_ADDR (SYS_DMA1+ 0)  /* (R/W) Base or current address              */
#define DMA_CTR  (SYS_DMA1+ 1)  /* (R/W) Base or current word count           */

#define SYS_PIC1        0x20    /* Base of primary interrupt controller */
#define PIC1_CMD  (SYS_PIC1+0)  /* (W) Command register (OCW2/OCW3)           */
#define PIC1_STAT (SYS_PIC1+0)  /* (R) Status register (ISR or IRR)           */
#define PIC1_MASK (SYS_PIC1+1)  /* (R/W) Interrupt mask register              */

#define SYS_PIC2        0xA0    /* Base of secondary int. controller          */
#define PIC2_CMD  (SYS_PIC2+0)  /* (W) Command register (OCW2/OCW3)           */
#define PIC2_STAT (SYS_PIC2+0)  /* (R) Status register (ISR or IRR)           */
#define PIC2_MASK (SYS_PIC2+1)  /* (R/W) Interrupt mask register              */

#define PICC_EOI        0x20    /* OCW2 (nonspecific) End-Of-Interrupt  */
                /*   command                                  */

/*
            Local Macros:
*/

#define PtrToLong(p) (((long)FP_SEG(p) << 4) + (long)FP_OFF(p))
```

```
                /* Macro to convert far pointer to      */
                /*   20-bit absolute address            */

#define DisableDma(ch) outp(DMA_WSMR, (ch)+4)   /* Disable DMA channel  */
#define EnableDma(ch)  outp(DMA_WSMR, (ch))     /* Enable DMA channel   */

/* Input and output from read-only command port, a shadow copy of the   */
/* port value is kept in MidGbl.CmpPort:                                 */

#define CmdIn()    (MidGbl.CmdPort)
#define CmdOut(val) (outp(MID_CMD, MidGbl.CmdPort = (int)(val)), \
                     outp(MID_CMD, MidGbl.CmdPort))

/* ------------------------------------------------------------------ */
/*               Global variables:                                     */
/* ------------------------------------------------------------------ */

MidAqGlobalType near MidGbl;    /* Global paramater/context variables   */

static int near DmaPageTable[8] = /* Table of DMA page register ports   */
            { 0x87, 0x83, 0x81, 0x82, -1, 0x8B, 0x89, 0x8A };

#define INTF_LENGTH 1024        /* Length of the interferogram */
#define PLIMIT 1024             /* number of points collected from Midac */
#define GH_LENGTH 512           /* Bytes in the global header */
#define SH_LENGTH 64            /* Bytes in the subfile header */
#define FEND 79                 /* set the key to terminate program */
#define DELAY_LENGTH 256

main(argc,argv)
int argc;
char *argv[];

{
  int MidAqInit(), MidAqSetGain();
  void MidAqStartScan();
  float raw_buf[INTF_LENGTH], intf_buf[INTF_LENGTH], flt_buf[SEG_LENGTH1];
  float flt_buf2[SEG_LENGTH2];
  float plinear2(), kalman();
  float delay[DELAY_LENGTH], dsc_result, kal_result=0.0;
  float delay2[DELAY_LENGTH], dsc_result2, kal_result2=0.0;
  int fburst();
  int scan=-1, index, burst, i, loop=0, igain=-1;
  char ch;
  void deriv(), rotate(), normal(), filter2(), lets_see_it();
  void logoega(), grf_results2();
  FILE *device, *fp2, *fp3;
  struct global_header gh;
  struct scan_header sh;
  long time_stamp();
  unsigned long ta0,ta1;
/*  long tm0,tm1,tm2,tm3,tm4,tm5,tm6,tm7,tm8; */
```

```
/*  int t0=0,t1=0,t2=0,t3=0,t4=0,t5=0,t6=0,t7=0,t8=0;   */

  if (argc != 3)
    {
    printf("\nUsage: mtrx outfile1 outfile2\n");
    exit(1);
    }

  /* identify the output device */
  device = stdout;
/* device = stdprn; */

  /* Open a file connection to the results */
  if ((fp2 = fopen(argv[1], "w")) == NULL)
    {
    printf("Unable to open \"%s\"\n", argv[1]);
    exit(2);
    }
  if ((fp3 = fopen(argv[2], "w")) == NULL)
    {
    printf("Unable to open \"%s\"\n", argv[2]);
    exit(3);
    }

  /* Zero-fill the delay line */
  for (i=0; i<DELAY_LENGTH; i++)
    delay[i] = 0.0;

  /* Set up the screen */
  _setvideomode(_ERESCOLOR);
  _setbkcolor(_BLUE);

  /* initialize the Midac interferometer */

    i = MidAqInit( -1, -1, igain, PLIMIT);
    if (i)
    {
            printf("Error: MidAqInit returned %d\n", i);
            return 1;
    }
/*    printf("MidCol initialized:\n");
    printf("  DMA Buffer at %Fp = %061X\n", MidGbl.DmaBuffer,
                PtrToLong(MidGbl.DmaBuffer)); */

/********************************************************************/
 /* check the instrument gain -- if too low, then increase gain
                               if too high, then decrease gain */
/*    igain++;
    MidAqStartScan();
    ta0 = (unsigned long)clock();
    while (!MidGbl.DmaDone)
     {
```

```
      tal = (unsigned long)clock();
      if ((tal-ta0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
        {
         printf("Error: Timeout on DMA completion\n");
         return 2;
        }
     }
    MidGbl.DmaActive = 0;
    for (index=0; index < PLIMIT; index++)
        raw_buf[index] = (float) MidGbl.DmaBuffer[index];
    burst = fburst(raw_buf,PLIMIT);
    while(fabs(raw_buf[burst]) <= 16384. && igain <= 7)
     {
      raw_buf[burst] *= 2.;
      igain++;
     }
      MidAqSetGain(igain);
      printf(".... setting the instrument A/D gain to = %d",igain);
      MidAqStartScan();
      ta0 = (unsigned long)clock();
      while (!MidGbl.DmaDone)
        {
          tal = (unsigned long)clock();
          if ((tal-ta0) > (unsigned long) (TIMEOUT * CLOCKS_PER_SEC))
            {
              printf("Error: Timeout on DMA completion\n");
              return 2;
             }
        }
       MidGbl.DmaActive = 0; */
/*******************************************************************/

/* set up the main loop to process data */

tloop:
    scan++;

/* check for the exit key */
    if (kbhit() != 0)
      {
       ch = getch();
       if (ch == FEND)
         {
          fclose (fp2);
          fclose (fp3);
          _setvideomode (_DEFAULTMODE);
          exit (1);
         }
       }

/* Collect 1 sample interferogram trace: */
```

```
    MidAqStartScan();
    ta0 = (unsigned long)clock();
    while ( !MidGbl.DmaDone )
    {
            ta1 = (unsigned long)clock();
            if ((ta1 - ta0) > (unsigned long)(TIMEOUT * CLOCKS_PER_SEC))
      {
                printf("Error: Timeout on DMA completion\n");
                exit(4);
      }
    }
    MidGbl.DmaActive = 0;

    /* convert the integer array to a ungain ranged floating array */
    for (index = 0; index < INTF_LENGTH; index++)
      raw_buf[index] =  (float) MidGbl.DmaBuffer[index];
/*    lets_see_it(device, "RAW", raw_buf, INTF_LENGTH); */


    /* Flip interferogram if burst is negative */
/*    tm0 = time_stamp(); */
    burst = fburst(raw_buf);
    if (raw_buf[burst] < 0.0)
      {
      for (i=0; i<INTF_LENGTH; i++)
        raw_buf[i] *= -1.0;
      }

    /* Calculate the derivative of the interferogram */
/*    tm1 = time_stamp(); */
    deriv(intf_buf, raw_buf);
/*    tm2 = time_stamp(); */
/*    lets_see_it(device, "DRV", intf_buf, INTF_LENGTH);  */

    /* find the burst of the interferogram */
    burst = fburst(intf_buf);
/*    tm3 = time_stamp(); */

    /* normalize the interferogram */
    normal(intf_buf);
/*    tm4 = time_stamp(); */
/*    lets_see_it(device, "NML", intf_buf, INTF_LENGTH);  */

    /* filter the short section */
    filter2(intf_buf, flt_buf, burst, 1);
    filter2(intf_buf, flt_buf2, burst, 2);
/*    tm5 = time_stamp(); */
/*    lets_see_it(device, "FLT", flt_buf, SEG_LENGTH);  */

    /* piece-wise linear discrimant */
    dsc_result = plinear2 (flt_buf, 1);
    dsc_result2 = plinear2 (flt_buf2, 2);
```

```
/*    tm6 = time_stamp(); */

    /* kalman filter */
    kal_result = kalman (scan, dsc_result, 1);
    kal_result2 = kalman (scan, dsc_result2, 2);
/*    tm7 = time_stamp(); */

    if (scan < DELAY_LENGTH)
      {
      delay[scan] = kal_result;
      delay2[scan] = kal_result2;
      }
    else
      {
      for (i=1; i<DELAY_LENGTH; i++)
        {
        delay[i-1] = delay[i];
        delay2[i-1] = delay2[i];
        }
      delay[DELAY_LENGTH-1] = kal_result;
      delay2[DELAY_LENGTH-1] = kal_result2;
      }

    loop = loop ^ 1;
    _setactivepage(loop);
    _clearscreen(_GCLEARSCREEN);
    _setvieworg(0,0);
    logoega(2,12);
    _setvieworg(64,175);
    grf_results2(scan, kal_result, kal_result2, delay, delay2);
    _setvisualpage(loop);
/*    tm8 = time_stamp(); */

    fprintf(fp2,"%04d  %10.5f\n", scan, kal_result);
    fprintf(fp3,"%04d  %10.5f\n", scan, kal_result);

    /* Update the timing totals */
/*    t0 += ((int) tm1 - tm0); */    /* burst/flip */
/*    t1 += ((int) tm2 - tm1); */    /* derivative */
/*    t2 += ((int) tm3 - tm2); */    /* burst location */
/*    t3 += ((int) tm4 - tm3); */    /* normalization */
/*    t4 += ((int) tm5 - tm4); */    /* filter */
/*    t5 += ((int) tm6 - tm5); */    /* discrimination */
/*    t6 += ((int) tm7 - tm6); */    /* kalman filter */
/*    t7 += ((int) tm8 - tm7); */    /* graphics */
/*    t8 += ((int) tm8 - tm0); */    /* total time */

    goto tloop;

}
/**************************** function logoega ******************************/
/*  logoega is a function used to create the CBDA logo for EGA graphics.
```

```
    The funtion requires two parameters, the x and y coordinates for the
    first letter "C". If the logo coordinates are outside the exceptable
    range, no logo will be plotted.

   author: John Ditillo
   modified by: Bob Kroutil

           logoega is based on the "old" CRDEC routine written by
           John T. Ditillo

   date: October 1992  */

void logoega(y,x)
int y, x;

{
  int xp, yp;

  if (y<23 & y>1 & x<76 & x>2)
    {

    /* draw the logo */
    _settextposition(y,x);
    _outtext ("C");

    _settextposition(y+1,x-1);
    _outtext ("B D");

    _settextposition(y+2,x);
    _outtext ("A");

    /* Calculate first pixel location */
    yp = y * 14 - 16;
    xp = x * 8 - 5;

    /* first benzene */
    _moveto(xp,yp);
    _lineto(xp-8,yp+3);
    _lineto(xp-8,yp+13);
    _lineto(xp,yp+17);
    _lineto(xp+8,yp+13);
    _lineto(xp+8,yp+3);
    _lineto(xp,yp);

    /* second benzene */
    _moveto(xp-8,yp+13);
    _lineto(xp-16,yp+17);
    _lineto(xp-16,yp+27);
    _lineto(xp-8,yp+31);
    _lineto(xp,yp+27);
    _lineto(xp,yp+17);
```

```
    /* third benzene */
    _moveto(xp+8,yp+13);
    _lineto(xp+16,yp+17);
    _lineto(xp+16,yp+27);
    _lineto(xp+8,yp+31);
    _lineto(xp,yp+27);

    /* fourth benzene */
    _moveto(xp-8,yp+31);
    _lineto(xp-8,yp+42);
    _lineto(xp,yp+45);
    _lineto(xp+8,yp+42);
    _lineto(xp+8,yp+31);

  }

}
/************************  function grf_results2 *****************************/
#define INIT_MAX .01

void grf_results2 (scan,kal,kal2,buf,buf2)
int scan;
float kal,kal2;
float buf[],buf2[];
{
  char buffer[80];
  int i, x, y, numpts, first_x, last_x, xscale;
  float yscale1, yscale2;
  static float max=INIT_MAX, max2=INIT_MAX;

  /* set the max value */
  if ((fabs((double)kal)) > max)
    max = (float) (fabs((double)kal));
  if ((fabs((double)kal2)) > max2)
     max2 = (float) (fabs((double)kal2));

  if (scan < DELAY_LENGTH)
    {
    numpts = scan;
    first_x = 0;
    last_x = DELAY_LENGTH-1;
    }
  else
    {
    numpts = DELAY_LENGTH;
    first_x = scan - (DELAY_LENGTH-1);
    last_x = scan;
    }

  /* Calculate the scaling factor */
  yscale1 = 75.0/max;
  yscale2 = 75.0/max2;
```

```
xscale = 512/DELAY_LENGTH;

_moveto (0,75);   /* Print the 2 z axis grids*/
_lineto (512,75);
_moveto (0,-75);
_lineto (512,-75);
_moveto (0,150);   /* Print the Y axis */
_lineto (0,2);
_moveto (0,-2);
_lineto (0,-150);

for(i = 0; i <= 512; i += 64)  /*Print the X axis tick marks */
  {
  _moveto(i, 80);
  _lineto(i, 75);
  _moveto(i, -70);
  _lineto(i, -75);
  }

  _moveto(-4, -150);      /* Print the y-axis tick marks */
  _lineto(0, -150);
  _moveto(-4, -2);
  _lineto(0, -2);
  _moveto(-4, 2);
  _lineto(0, 2);
  _moveto(-4, 150);
  _lineto(0, 150);

/* label the axis */
sprintf(buffer,"%04d", first_x);
_settextposition(25,7);
_outtext(buffer);

 sprintf(buffer,"%04d", last_x);
_settextposition(25,70);
_outtext(buffer);

sprintf(buffer,"% 5.4f", max2);
_settextposition(3,0);
_outtext(buffer);

sprintf(buffer,"% 5.4f", -max2);
_settextposition(12,0);
_outtext(buffer);

sprintf(buffer,"% 5.4f", max);
_settextposition(14,0);
_outtext(buffer;;

sprintf(buffer,"% 5.4f", -max);
_settextposition(24,0);
_outtext(buffer);
```

```
  sprintf(buffer,"SCAN: %5d                : % 7.5f", scan, kal2);
  for (i=0; i < 10; i++)
    buffer[i+13] = hdmsg2[i];
  _settextposition(1,27);
  _outtext(buffer);

  sprintf(buffer,"SCAN: %5d                : % 7.5f", scan, kal);
  for (i=0; i < 10; i++)
    buffer[i+13] = hdmsg1[i];
  _settextposition(13,27);
  _outtext(buffer);

  sprintf(buffer,"End key to exit");
  _settextposition(25,35);
  _outtext(buffer);

  /* plot the data */
  _moveto (0, (int) -(buf[0] * yscale1 - 75));
  for (i=1; i < numpts; i++)
    {
     x = i * xscale;
     y = (int) -(buf[i] * yscale1 - 75);
     _lineto (x,y);
    }
   _moveto (0, (int) -(buf2[0] * yscale2 + 75));
   for (i=1; i < numpts; i++)
    {
     x = i * xscale;
     y = (int) -(buf2[i] * yscale2 + 75);
     _lineto (x,y);
     }
}
/*************************** function fburst *************************/
int fburst(buffer)
float buffer[];
{

/*  int index, bloc;
  double bval;

  bloc = 0;
  bval = (double) buffer[0];

  for (index = 1; index < INTF_LENGTH; index++)
    if (fabs((double) buffer[index]) > bval)
      {
      bval = fabs((double) buffer[index]);
      bloc = index;
      }

  return (bloc);  */
```

```
  int i, max_loc, min_loc;
  float max_val=0.0, min_val=0.0;

  for (i=0; i<INTF_LENGTH; i++)
    if (buffer[i] > max_val)
      {
      max_val = buffer[i];
      max_loc = i;
      }
    else if (buffer[i] < min_val)
      {
      min_val = buffer[i];
      min_loc = i;
      }

  if (fabs((double) min_val) > max_val)
    return(min_loc);
  else
    return(max_loc);

}
/**************************** function deriv ******************************/
void deriv(buf1, buf2)
float buf1[], buf2[];
{

  int i2n,in,ib,i2b;
  int index, isrt, ifin, ncent;
  float denom;

  /* use the forward difference for the first two points */
  denom = 2.0;
  i2n = 2;
  in = 1;
  for (index=0; index < 2; index++, i2n++, in++)
    buf1[index] = (-buf2[i2n] + 4.0*buf2[in] - 3.0*buf2[index])/denom;


  /* use the backward difference for the last two points */
  i2b = INTF_LENGTH - 4;
  ib = INTF_LENGTH - 3;
  isrt = INTF_LENGTH - 2;
  for (index=isrt; index < INTF_LENGTH; index++, i2b++, ib++)
    buf1[index] = (buf2[i2b] - 4.0*buf2[ib] +3.0*buf2[index])/denom;


  /* use the central difference for the middle points */
  ncent = INTF_LENGTH - 5;
  isrt = 2;
  ifin = INTF_LENGTH - 2;
  i2b = 0;
  ib = 1;
```

```
  in = 3;
  i2n = 4;
  denom = 12.0;
  for (index=isrt; index < ifin; index++, i2n++, in++, ib++, i2b++)
     buf1[index] = (buf2[i2b] - 8.0*buf2[ib] + 8.0*buf2[in] -
buf2[i2n])/denom;


}
/***************************** function normal ******************************/
void normal(buffer)
float buffer[];
{

  int index;
  float ssq = 0.0;

  for (index=0; index < INTF_LENGTH; index++)
    ssq += buffer[index] * buffer[index];

  if (ssq > 0.0)
    ssq = INTF_LENGTH / sqrt(ssq);
  else
    ssq = 1.0;

  for (index=0; index < INTF_LENGTH; index++)
    buffer[index] *= ssq;

}
/************************* function filter2 ********************************/
void filter2(in_buf, out_buf, burst, ifilt)
float in_buf[];
float out_buf[];
int burst,ifilt;
{

  int i, j, k;

  if (ifilt == 1)
    {

/* do the first digital filter */
  for (i=0, k=SEG_OFFSET1+burst-1; i<SEG_LENGTH1; i++, k++)
    {
    out_buf[i] = flt_intercepts1[i];
    for (j=0; j < flt_length1[i]; j++)
      out_buf[i] += flt_coefs1[i][j] * in_buf[ k+flt_offsets1[i][j] ];
    }
   }
    else
   {
/* do the second digital filter */
```

```
  for (i=0, k=SEG_OFFSET2+burst-1; i<SEG_LENGTH2; i++, k++)
    {
    out_buf[i] = flt_intercepts2[i];
    for (j=0; j < flt_length2[i]; j++)
      out_buf[i] += flt_coefs2[i][j] * in_buf[ k+flt_offsets2[i][j] ];
    }
   }
}
/************************* function plinear2 ******************************/
float plinear2(in_buf, ifilt)
float in_buf[];
int ifilt;
{

  float dsc_max=-100.0;
  float dsc;
  int i, j, k;

  if (ifilt == 1)
  {
  for (i=0; i < DSC_PASS1; i++)
    {
    dsc = dsc_intercepts1[i];
    for (j=0; j < SEG_LENGTH1; j++)
      dsc += in_buf[j] * dsc_coefs1[i][j];

    if (dsc > dsc_max)
      dsc_max = dsc;
    }
   }
    if(ifilt != 1)
    {
     for (i=0; i < DSC_PASS2; i++)
      {
       dsc = dsc_intercepts2[i];
       for (j=0; j < SEG_LENGTH2; j++)
        dsc += in_buf[j] * dsc_coefs2[i][j];

     if (dsc > dsc_max)
       dsc_max = dsc;
      }
    }

  return(dsc_max);
}
/************************** function kalman ********************************/
float kalman(scan_num, in_value, k)
int scan_num,k;
float in_value;
{
  static float sum[2] = {0.0,0.0};
  static float sumsq[2]= {0.0,0.0};
```

```
static float q[2] = {0.0,0.0};
static float clkip[2] = {0.0,0.0};
static float sigcl2[2];
static float skip[2];
static float prev_input[2*KAL_WIN+1];
static float beta[2*KAL_WIN+1];
static float prev_input2[2*KAL_WIN+1];
static float beta2[2*KAL_WIN+1];
int nm, i, j;
float temp, sbase, skm, kal_gain, kal_result, clcov, clkm;
char bl;

nm = 2 * KAL_WIN + 1;

if (scan_num == 0)
  {
  /* setup info for the kalman */
  temp = 3.0/(float)(KAL_WIN*(KAL_WIN+1)*(2*KAL_WIN+1));
  for (i=-KAL_WIN, j=0; i<KAL_WIN+1; i++, j++)
    {
    if (k == 1)
     beta[j] = temp * (float) i;
    else
     beta2[j] = temp * (float) i;
    }
  }

if (scan_num < KAL_SETUP)
  {
  sum[k] += in_value;
  sumsq[k] += in_value * in_value;

  return(0.0);
  }

else if (scan_num == KAL_SETUP)
  {
  sbase = (float) KAL_SETUP;
  sigcl2[k] = (sbase * sumsq[k] - sum[k] * sum[k])/(sbase*(sbase-1.0));
  skip[k] = sigcl2[k];
  return(0.0);
  }

else
  {
  clkm = clkip[k];
  skm = skip[k] + q[k];

  /* compute the kalman gain */
  kal_gain = skm / (skm + sigcl2[k]);

  /* update the intensity covariance */
```

```
    clcov = (1.0 - kal_gain) * skm;

    /* update the intensity estimate */
    kal_result = clkm + kal_gain * (in_value - clkm);

    /* update array of previous values */
    if (k == 1)
    {
    for (i=0; i<nm-1; i++)
      prev_input[i] = prev_input[i+1];
    prev_input[nm-1] = in_value;
/*    printf("\n in_value=%5.4f ",in_value); */

    /* update the Q estimate and compute the moving average */
    sum[k] = 0.0;
    for (i=0; i<nm; i++)
      sum[k] += beta[i] * prev_input[i];
/*       printf("\ni=%d k=%d prev=%5.4f",i,k,prev_input[i][k]); */
    }
    else
    {
    for (i=0; i<nm-1; i++)
      prev_input2[i] = prev_input2[i+1];
    prev_input2[nm-1] = in_value;

    /* update the Q estimate and compute the moving average */
    sum[k] = 0.0;
    for (i=0; i<nm; i++)
       sum[k] += beta2[i] * prev_input2[i];
    }

    q[k] = sum[k] * sum[k];
    clkip[k] = kal_result;
    skip[k] = clcov;
/*    printf("\nq[1]=%5.4f q[2]=%5.4f",q[1],q[2]);
    printf("\nclkip[1]=%5.4f clkip[2]=%5.4f",clkip[1],clkip[2]);
    printf("\nskip[1]=%5.4f skip[2]=%5.4f",skip[1],skip[2]);
    scanf("%s",b1); */
    return(kal_result);
    }
}
/************************** function lets_see_it ****************************/
void lets_see_it(device, label, buffer, length)
FILE *device;
char label[];
float buffer[];
int length;
{
  int i;

  if (device == stdout)        /* Output to display */
    {
```

```
    fprintf(device,"\n\n\n\n");
    for (i=0; i < length; i += 2)
    fprintf(device, "%s# %4d = %12.3f      %s# %4d = %12.3f\n",
            label, i+1, buffer[i], label, i+2, buffer[i+1]);
    }
  else                          /* Output to the printer */
    {
    fprintf(device,"\r\n\r\n\r\n\r\n");
    for (i=0; i < length; i += 4)
      {
      fprintf(device, "%s# %4d = %12.3f      %s# %4d = %12.3f    ",
              label, i+1, buffer[i], label, i+2, buffer[i+1]);
      fprintf(device, "%s# %4d = %12.3f      %s# %4d = %12.3f\r\n",
              label, i+3, buffer[i+2], label, i+4, buffer[i+3]);
      }
    }
}
/************************* function time_stamp ******************************/
long time_stamp()
  {
  union REGS regs;                /*SETUP FOR REGISTER USE*/
  long tc;

  regs.h.ah = 0;                  /*SET ACC FOR TIME TYPE INTERRUPT*/
  int86( 0x1a, &regs, &regs );    /*GENERATE INTERRUPT FOR TIME*/
  tc = (((long) regs.x.cx) << 16) + regs.x.dx;
  return(tc);                     /*RETURN CLOCK TICK*/
  }
/* ------------------------------------------------------------------- */
/*      in:     Allow port input during debug.                         */
/*              This is necessary for CV 4.00--the "I" command (port   */
/*              input is broken. The circumvention is to include a     */
/*              a global function such as in() below, trace at least   */
/*              as far as the main() function, then "?in(port)" or     */
/*              "?in(port),x" to read port contents.                   */
/* ------------------------------------------------------------------- */

int in( unsigned port )
{
    int i;
    i = inp(port);
    return i;

} /* in */


/* ------------------------------------------------------------------- */
/*      IoDelay:        I/O delay for IBM/AT and clones.               */
/*                                                                     */
/*      This dummy function is used to generate a few clocks of delay */
/*      between consecutive accesses to certain I/O ports. Basically,  */
/*      the call/return sequence is more than enough. Assembler        */
```

```
/*      programs typically use a "JMP SHORT $+2" instruction, but        */
/*      the MSC7 inline assembler doesn't seem to handle the "$"         */
/*      token very well. The delay is necessary on IBM AT machines       */
/*      and true compatibles.                                            */
/*                                                                       */
/*      Needless to say, allowing this function to be inlined would      */
/*      be a bad idea...                                                 */
/* --------------------------------------------------------------------- */

static void near IoDelay(void)
{
   ;
} /* IoDelay */

/* --------------------------------------------------------------------- */
/*      GetDmaBuffer:   Allocate a byte-DMA compatible buffer            */
/*                                                                       */
/*      A byte DMA buffer cannot cross a 64K-byte absolute address       */
/*      boundary.                                                        */
/*                                                                       */
/*      Returns pointer to buffer if successful, NULL otherwise.         */
/* --------------------------------------------------------------------- */

void far *GetDmaBuffer(long Size)
{
   #define MaxTries 16          /* Maximum attempts before failure       */

   void         far *failed[MaxTries],
                 far *try,
                 far *retry;
   unsigned     begoff, endoff;
   int          i, nfail=0;

   if (Size>MAXDMA || Size<=0) return NULL;

   for (;;)                               /* Repeat until explicit break: */
   {
         try = malloc((size_t)Size);
         if ( try==NULL ) break;

/* Test for 64K block wraparound:                                        */

         begoff = (FP_SEG(try) << 4) + FP_OFF(try);
         endoff = begoff + (unsigned)Size - 1;
         if (endoff >= begoff) break;     /* Success if all in 1 block      */

/* Current attempt crosses boundary, retry if failed list not full:      */

         if (nfail == MaxTries)
         {
             free(try);
             try = NULL;
```

```
                break;
            }

/* Resize current try to end on 64K absolute boundary and add it to      */
/* the failed list:                                                      */

            retry = realloc(try, 1+~begoff);
            if ( retry != NULL )
                try = retry;
            failed[nfail++] = try;
    }

/* Arrive here via explicit break. Free failed attempt pointers, if      */
/* any and exit. The try variable has been set to a pointer on success   */
/* or to NULL on error.                                                  */

    for( i=0; i<nfail; ++i )
    {
            free( failed[i] );
    }

    return try;

#undef MaxTries                         /* Undefine "local" macros       */

} /* GetDmaBuffer */


/* ------------------------------------------------------------------- */
/*      StartDma:       Start a DMA operation.                           */
/*                                                                       */
/*      This is a cut-down version to do input only, specifically        */
/*      using DMA info in MidGbl structure.                              */
/* ------------------------------------------------------------------- */

void StartDma(void)
{
    long        addr = PtrToLong(MidGbl.DmaBuffer);
    int         size = (int)MidGbl.DmaSize;
    unsigned    ch   = 2*MidGbl.DmaChannel;

    DisableDma(MidGbl.DmaChannel);
    IoDelay();                          /* Wait a few CPU clocks         */
    outp(DMA_MODE, 0x44+MidGbl.DmaChannel);
                /* DMA Mode: single-block,      */
                /*   increment address,         */
                /*   no autoinitialize,         */
                /*   "write transfer" -> cpu    */
    IoDelay();                          /* Wait a few CPU clocks         */

    outp(DMA_CLRF,0);                   /* Set to receive LSB first      */
    IoDelay();                          /* Wait a few CPU clocks         */
```

```
    outp(DMA_CTR+ch, (int)size);        /* Send byte count            */
    IoDelay();                          /* Wait a few CPU clocks      */
    outp(DMA_CTR+ch, (int)size >> 8);
    IoDelay();                          /* Wait a few CPU clocks      */

    outp(DMA_ADDR+ch, (int)addr);       /* Send address               */
    IoDelay();                          /* Wait a few CPU clocks      */
    outp(DMA_ADDR+ch, (int)addr >> 8);
    IoDelay();                          /* Wait a few CPU clocks      */

    outp(MidGbl.DmaPageReg, (int)(addr>>16));
                /* Set page reg to top 8 bits   */
    IoDelay();                          /* Wait a few CPU clocks      */

    EnableDma(MidGbl.DmaChannel);       /* Finally, enable DMA        */

} /* StartDma */


/* ---------------------------------------------------------------- */
/*      SetIrqEnable:   Set/Reset IRQ enable status for specified    */
/*                      channel.                                     */
/*                                                                   */
/*      Please note that the sense of the "Enable" argument is a C-  */
/*      style boolean. Nonzero, or "true", enables the channel. This */
/*      is opposite from the 8259 mask register, where a 1 disables  */
/*      the channel and 0 enables.                                   */
/* ---------------------------------------------------------------- */

void SetIrqEnable(
    int         IrqNumber,      /* Interrupt channel, 0-15           */
    int         Enable)         /* New enable status for this channel */
                /*   0 = disable interrupts           */
                /*   nonzero = enable interrupts      */
{
    unsigned    port;
    int         mask, val;

    if (IrqNumber < 8)
    {
        port = PIC1_MASK;               /* Primary 8259 port            */
        mask = 1 << IrqNumber;
    }
    else
    {
        port = PIC2_MASK;               /* Secondary 8259 port          */
        mask = 1 << (IrqNumber-8);
    }

    val = inp(port) | mask;             /* Set to mask disable        */
    if (Enable) val -= mask;            /* Set to enable if requested */
    outp(port, val);                    /* Update port                */
```

```
} /* SetIrqEnable */


/* -------------------------------------------------------------- */
/*      MidAqStartScan: Start new data collect operation           */
/*                                                                 */
/*      This is a skeleton of what is needed to begin a new data   */
/*      scan, or series of accumulated scans, on the Midac FT-IR.  */
/* -------------------------------------------------------------- */

void MidAqStartScan(void)
{
    SetIrqEnable(MidGbl.IrqNum, 0);     /* Disable interrupt channel    */
    IoDelay();                          /* Wait a few CPU clocks        */
    DisableDma(MidGbl.DmaChannel);      /* Disable DMA channel          */
    IoDelay();                          /* Wait a few CPU clocks        */

    StartDma();                         /* Start DMA channel            */

    SetIrqEnable(MidGbl.IrqNum, 1);     /* Enable interrupt channel     */

/* Set gain and retrace interferometer:                                */

    CmdOut( MidGbl.GainPort | MIDC_EOS | MIDC_IRQ );
                /* Start IRQ clear pulse*/
    IoDelay();                                  /* Wait a few CPU clocks*/
    CmdOut( CmdIn() &~(MIDC_EOS + MIDC_IRQ) );  /* End IRQ clear pulse, */
                /* Start retrace pulse  */
    IoDelay();                                  /* Wait a few CPU clocks*/
    while (inp(MID_STAT) & MIDS_FLYBK);         /* Wait for turnaround  */
    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ));   /* End retrace pulse    */
    IoDelay();                                  /* Wait a few CPU clocks*/

    /* Note: May need to insert delay here, 10-20ms, to allow for      */
    /* hardware bug in Midac interface causing early DMA requests.     */
                _asm xor  cx,cx
         here:   _asm loop here

    MidGbl.DmaActive = 1;               /* Set global DMA status flags  */
    MidGbl.DmaDone = 0;

    CmdOut( CmdIn() | MIDC_DMA );       /* Enable DMA at interface      */

} /* MidAqStartScan */


/* -------------------------------------------------------------- */
/*      MidAqDmaDone:   Interrupt Handler for DMA completion       */
/*                                                                 */
/*      This version simply notes DMA completion, retraces the     */
/*      interferometer, and disables DMA at both the 8237 and at   */
/*      the Midac interface board.  This would be the natural place */
```

```
/*      to insert co-add logic for averaging interferograms.            */
/* ---------------------------------------------------------------- */

void _cdecl _interrupt far MidAqDmaDone(void)
{
    MidGbl.DmaDone = 1;                  /* Note DMA completion        */

    CmdOut( CmdIn() &~MIDC_DMA );        /* Disable DMA at interface   */
    DisableDma(MidGbl.DmaChannel);       /*  then disable channel      */
    IoDelay();                           /* Wait a few CPU clocks      */

/* Retrace interferometer:                                             */

    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ) );  /* Start IRQ clear pulse*/
    CmdOut( CmdIn() &~(MIDC_EOS + MIDC_IRQ) );  /* End IRQ clear pulse, */
                /* Start retrace pulse  */
    _enable();                                  /* Interrupts on now    */
    while (inp(MID_STAT) & MIDS_FLYBK);         /* Wait for turnaround  */
    CmdOut( CmdIn() | (MIDC_EOS + MIDC_IRQ));   /* End retrace pulse    */

    /* This is the place to put co-add logic and possibly start the    */
    /* DMA controller for a new scan. Note that the instrument will    */
    /* scan anyway--the decision is whether or not to collect the data. */

    /* Note: May need to insert delay, 10-20ms, to allow for           */
    /* hardware bug in Midac interface, if another scan is to be       */
    /* started here.                                                   */

    outp(PIC1_CMD, PICC_EOI);            /* Issue EOI to master        */
    IoDelay();                           /* Wait a few CPU clocks      */
    if (MidGbl.IrqNum > 7)               /* If interrupt is on slave   */
        outp(PIC2_CMD, PICC_EOI);            /*   then issue secondary EOI   */

} /* MidAqDmaDone */


/* ---------------------------------------------------------------- */
/*      MidAqSetGain:   Set Signal Gain                                 */
/*                                                                      */
/* ---------------------------------------------------------------- */

int MidAqSetGain(int SignalGain)
{
    int gainport = ((~SignalGain << MIDC_GSHIFT) & MIDC_GMASK);
    int oldgain = MidGbl.GainVal;

    if (SignalGain<0 || SignalGain>7)
        return -1;

    CmdOut(gainport | (CmdIn() & ~MIDC_GMASK));
    MidGbl.GainVal = SignalGain;
    MidGbl.GainPort = gainport;
```

```
    return oldgain;

} /* MidAqSetGain */


/* ---------------------------------------------------------------------- */
/*                                                                        */
/*      MidAqTerm:      Data collect termination                          */
/*                                                                        */
/*      This function is not explicitly called, but is called at          */
/*      program termination via the atexit() facility. The primary        */
/*      task is to disable DMA and the terminal count interrupt and       */
/*      restore the IRQ vector.                                           */
/* ---------------------------------------------------------------------- */

void MidAqTerm(void)
{

    SetIrqEnable(MidGbl.IrqNum, 0);      /* Disable interrupt channel     */
    DisableDma(MidGbl.DmaChannel);       /* Disable DMA channel           */
    CmdOut(MIDC_EOS);                    /* Reset the interferometer      */
    IoDelay();                           /* Wait a few CPU clocks         */

    if (MidGbl.OldIrqVec != NULL)
    {
            _dos_setvect(MidGbl.IrqVecNo, MidGbl.OldIrqVec);
            MidGbl.OldIrqVec = NULL;
    }

} /* MidAqTerm */


/* ---------------------------------------------------------------------- */
/*      MidAqInit:      Initialize Midac interface for data collect       */
/*                                                                        */
/*      The arguments to this function provide for setup parameters       */
/*      and/or nonstandard interface board configurations. Each is        */
/*      either a nonnegative integer value, or -1 to use the              */
/*      predefined default value.                                         */
/*                                                                        */
/*      The first two arguments (DmaChannel, IrqNumber) describe the      */
/*      configuration of the Midac interface board. Current interface     */
/*      boards are hardwired for DMA channel 1 and are jumper             */
/*      selectable to use either IRQ2 or IRQ3. Other options could        */
/*      conceivably be possible for unusual custom requirements.          */
/*      In general, however, such a modified interface board would        */
/*      be incompatible with existing SpectraCalc and LabCalc drivers.    */
/*                                                                        */
/*      The buffer size argument (MaxPoints) is necessary to allocate     */
/*      a DMA buffer. This buffer has the hardware-enforced               */
/*      requirement to not cross a 64K-byte absolute memory boundary.     */
/*      This is the strictest dynamic allocation requirement in a         */
```

```
/*      typical data collect application, and should be done first.       */
/*      If co-addition of interferograms is to be performed, this is      */
/*      might be a good place to allocate an accumulator buffer as        */
/*      well.                                                             */
/*                                                                        */
/*      The gain argument (SignalGain) provides the initial signal        */
/*      gain level for programming the interface. This value is           */
/*      subject to change during program operation, but some initial      */
/*      value is required.                                                */
/* ---------------------------------------------------------------------- */

int MidAqInit(
    int         DmaChannel,     /* DMA channel number, 0-3                */
    int         IrqNumber,      /* PC/ISA interrupt channel number        */
    int         SignalGain,     /* Signal gain level, 0-7                 */
    int         MaxPoints)      /* Max data points in collect buffer      */
{
    int         i, dmachan, irqnum, maxpts, gainval, gainport;

/* Translate and validate input paramters...                              */

    dmachan     = DmaChannel>=0 ? DmaChannel : DMA;
    irqnum      = IrqNumber >=0 ? IrqNumber  : IRQ;
    gainval     = SignalGain>=0 ? SignalGain : GAIN;
    maxpts      = MaxPoints>=0  ? MaxPoints  : BUFPTS;

    if (dmachan != DMA) return -1;      /* ***temp*** need to know page */
                /* register addresses for other */
                /* DMA channels to generalize   */
                /* this for other byte channels */

    if (dmachan<0 || dmachan>3)
            return -1;
    if (irqnum<0 || irqnum>15)
            return -1;
    if (gainval<0 || gainval>7)
            return -1;
    if (maxpts<1 || maxpts>(MAXDMA / 2))
            return -1;

/* Bring the hardware interface to idle state:                            */

    gainport = (~gainval << MIDC_GSHIFT) & MIDC_GMASK;
                /* Compute inverted gain val    */
    MidGbl.GainVal      = gainval;      /* Save requested gain            */
    MidGbl.GainPort     = gainport;     /* Save port image                */

    CmdOut(gainport | MIDC_EOS);        /* Set gain, DMA off, and         */
                /*   EOS,IRQ strobes off.       */

    SetIrqEnable(irqnum, 0);            /* Disable interrupt channel      */
    DisableDma(dmachan);                /* Disable DMA channel            */
```

```
    IoDelay();                              /* Wait a few CPU clocks          */

/* Initialize DMA:                                                            */

    MidGbl.DmaDone       = 0;
    MidGbl.DmaActive     = 0;
    MidGbl.MaxPoints     = maxpts;
    MidGbl.DmaChannel    = dmachan;
    MidGbl.DmaPageReg    = DmaPageTable[dmachan];
    MidGbl.DmaSize       = (long)maxpts * sizeof(unsigned short);
    MidGbl.DmaBuffer     = GetDmaBuffer(MidGbl.DmaSize);
    if (MidGbl.DmaBuffer == NULL)
            return -1;

    for (i=0; i<maxpts; ++i)                /* Put recognizable null data     */
            MidGbl.DmaBuffer[i] = 0xEEEE;   /*   in buffer for debug            */

/* Initialize IRQ channel                                                     */

    MidGbl.IrqNum        = irqnum;
    MidGbl.IrqVecNo      = (irqnum<8 ? 0x08 : 0x68) + irqnum;
    MidGbl.OldIrqVec     = _dos_getvect(MidGbl.IrqVecNo);
    _dos_setvect(MidGbl.IrqVecNo, MidAqDmaDone);

    atexit(MidAqTerm);

    return 0;

} /* MidAqInit */
```

Blank

APPENDIX J

DISK DATA READ PATTERN RECOGNITION PROGRAM

```
/*********************************************************************/
/*
    Program MTRXD

    This program is a "C" version of the TESTWV program located
    on the Silicon Graphics computer.  This program will process
    interferograms collected on disk and display the result
    of the filtering and pattern recognition.

    author of modified C version:  Bob Kroutil

    date: October 1992   */

/********************************************************************/

#include <stdio.h>
#include <fcntl.h>
#include <math.h>
#include <bios.h>
#include <graph.h>
#include <dos.h>
#include "headers.def"
#include "mtrx.def"
#include "filter1.inc"  /* include the filter coefficients */
#include "discrim1.inc" /* include the pattern recognition coefficients */

#define INTF_LENGTH 1024        /* Length of the interferogram */
#define GH_LENGTH 512           /* Bytes in the global header */
#define SH_LENGTH 64            /* Bytes in the subfile header */
#define FEND 79
#define DELAY_LENGTH 256

main(argc,argv)
int argc;
char *argv[];

{
  float raw_buf[INTF_LENGTH], intf_buf[INTF_LENGTH], flt_buf[SEG_LENGTH1];
  float plinear(), kalman();
  float delay[DELAY_LENGTH], dsc_result, kal_result=0.0;
  int fburst();
  int raw_data[INTF_LENGTH];
  int scan, index, burst, i, loop=0;
  int fp1;
  char ch;
  void deriv(), rotate(), normal(), filter(), lets_see_it();
  void logoega(), grf_results();
  FILE *device, *fp2;
  struct global_header gh;
  struct scan_header sh;
  long time_stamp();
  long tm0,tm1,tm2,tm3,tm4,tm5,tm6,tm7,tm8;
```

```
int t0=0,t1=0,t2=0,t3=0,t4=0,t5=0,t6=0,t7=0,t8=0;
union REGS inregs;   /* REG structure for timing input */
union REGS outregs;  /* REG structure for timing output */

if (argc != 3)
  {
  printf("\nUsage: mtrxd infile outfile\n");
  exit(1);
  }

/* prompt user for the output device */
/*  printf("Enter the desired output device for intermediate results\n");
 printf("(S)creen or (P)rinter >> ");
 ch = getchar();
 while (getchar() !='\n');
 if (ch == 'P' | ch =='p')
   device = stdprn;
 else
   device = stdout; */
 device = stdout;


 /* Open a file connection to the Midac data file */
 if ((fp1 = open(argv[1], O_RDONLY|O_BINARY)) < 0)
   {
   printf("\n\"mtrxd\" is unable to open %s\n",argv[1]);
   exit(1);
   }

 /* Open a file connection to the results */
 if ((fp2 = fopen(argv[2], "w")) == NULL)
   {
   printf("Unable to open \"%s\"\n", argv[2]);
   exit(1);
   }
 else
   fprintf(fp2,"%s\n", argv[1]);

 /* Zero-fill the delay line */
 for (i=0; i<DELAY_LENGTH; i++)
   delay[i] = 0.0;

 /* Set up the screen */
 _setvideomode(_ERESCOLOR);
 _setbkcolor(_BLUE);

 /* read in the global header */
 read(fp1, &gh, GH_LENGTH);

 for (scan = 0; scan < gh.stop_scan; scan++)
   {
```

```
    /* if using a 486 computer then delay each calculation for
       display purposes -- remove this section for 386 version */
     inregs.h.ah = 0x86;     /* delay service */
     inregs.x.cx = 5;        /* set high order delay word */
     inregs.x.dx = 0;        /* set low order delay word */
     int86 (0x15,&inregs,&outregs); /* call to ROM BIOS timer delay service */

    /* Check for exit key */
    if (kbhit() != 0)
      {
      ch = getch();
      if (ch == FEND)
        {
        fclose(fp2);
        close(fp1);
        _setvideomode(_DEFAULTMODE);
        exit(1);
        }
      }

    read(fp1, &sh, SH_LENGTH);                /* read the subfile header */

    read(fp1, raw_data, INTF_LENGTH*2);       /* read the subfile data */

    /* convert the integer array to a ungain ranged floating array */
    for (index = 0; index < INTF_LENGTH; index++)
      raw_buf[index] =  (float) raw_data[index];
/*    lets_see_it(device, "RAW", raw_buf, INTF_LENGTH); */


    /* Flip interferogram if burst is negative */
    tm0 = time_stamp();
    burst = fburst(raw_buf);
    if (raw_buf[burst] < 0.0)
      for (i=0; i<INTF_LENGTH; i++)
        raw_buf[i] *= -1.0;

    /* Calculate the derivative of the interferogram */
    tm1 = time_stamp();
    deriv(intf_buf, raw_buf);
    tm2 = time_stamp();
/*    lets_see_it(device, "DRV", intf_buf, INTF_LENGTH); */

    /* find the burst of the interferogram */
    burst = fburst(intf_buf);
    tm3 = time_stamp();

    /* normalize the interferogram */
    normal(intf_buf);
    tm4 = time_stamp();
/*    lets_see_it(device, "NML", intf_buf, INTF_LENGTH); */
```

```
    /* filter the short section */
    filter(intf_buf, flt_buf, burst);
    tm5 = time_stamp();
/*    lets_see_it(device, "FLT", flt_buf, SEG_LENGTH); */

    /* piece-wise linear discrimant */
    dsc_result = plinear(flt_buf);
    tm6 = time_stamp();

    /* kalman filter */
    kal_result = kalman(scan, dsc_result);
    tm7 = time_stamp();

    if (scan < DELAY_LENGTH)
      delay[scan] = kal_result;
    else
      {
      for (i=1; i<DELAY_LENGTH; i++)
        delay[i-1] = delay[i];
      delay[DELAY_LENGTH-1] = kal_result;
      }

    loop = loop ^ 1;
    _setactivepage(loop);
    _clearscreen(_GCLEARSCREEN);
    _setvieworg(0,0);
    logoega(2,12);
    _setvieworg(64,175);
    grf_results(scan, kal_result, delay);
    _setvisualpage(loop);
    tm8 = time_stamp();

    fprintf(fp2,"%04d  %10.5f\n", scan, kal_result);

    /* Update the timing totals */
    t0 += ((int) tm1 - tm0);     /* burst/flip */
    t1 += ((int) tm2 - tm1);     /* derivative */
    t2 += ((int) tm3 - tm2);     /* burst location */
    t3 += ((int) tm4 - tm3);     /* normalization */
    t4 += ((int) tm5 - tm4);     /* filter */
    t5 += ((int) tm6 - tm5);     /* discrimination */
    t6 += ((int) tm7 - tm6);     /* kalman filter */
    t7 += ((int) tm8 - tm7);     /* graphics */
    t8 += ((int) tm8 - tm0);     /* total time */
    }
  close(fp1);
  fclose(fp2);
  _setvideomode(_DEFAULTMODE);

  printf("\n\n");
  printf("Burst/Flip:      %02d\n", t0/scan);
  printf("Derivative:      %02d\n", t1/scan);
```

```
  printf("Burst location:  %02d\n", t2/scan);
  printf("Normalization:   %02d\n", t3/scan);
  printf("Filter:          %02d\n", t4/scan);
  printf("Discrimination:  %02d\n", t5/scan);
  printf("Kalman:          %02d\n", t6/scan);
  printf("Graphics:        %02d\n", t7/scan);
  printf("                ====\n");
  printf("Total time:      %02d ticks or  %d mseconds\n",
          t8/scan, (t8/scan)*55);

}
/**************************** function logoega *****************************/
/*  logoega is a function used to create the CBDA logo for EGA graphics.
    The funtion requires two parameters, the x and y coordinates for the
    first letter "C". If the logo coordinates are outside the exceptable
    range, no logo will be plotted.

   author: John Ditillo
   modified by: Bob Kroutil

           logoega is based on the "old" CRDEC logo program
           written by John T. Ditillo

   date: October 1992  */

void logoega(y,x)
int y, x;

{
  int xp, yp;

  if (y<23 & y>1 & x<76 & x>2)
    {

    /* draw the logo */
    _settextposition(y,x);
    _outtext ("C");

    _settextposition(y+1,x-1);
    _outtext ("B D");

    _settextposition(y+2,x);
    _outtext ("A");

    /* Calculate first pixel location */
    yp = y * 14 - 16;
    xp = x * 8 - 5;

    /* first benzene */
    _moveto(xp,yp);
    _lineto(xp-8,yp+3);
    _lineto(xp-8,yp+13);
```

```
    _lineto(xp,yp+17);
    _lineto(xp+8,yp+13);
    _lineto(xp+8,yp+3);
    _lineto(xp,yp);

    /* second benzene */
    _moveto(xp-8,yp+13);
    _lineto(xp-16,yp+17);
    _lineto(xp-16,yp+27);
    _lineto(xp-8,yp+31);
    _lineto(xp,yp+27);
    _lineto(xp,yp+17);

    /* third benzene */
    _moveto(xp+8,yp+13);
    _lineto(xp+16,yp+17);
    _lineto(xp+16,yp+27);
    _lineto(xp+8,yp+31);
    _lineto(xp,yp+27);

    /* fourth benzene */
    _moveto(xp-8,yp+31);
    _lineto(xp-8,yp+42);
    _lineto(xp,yp+45);
    _lineto(xp+8,yp+42);
    _lineto(xp+8,yp+31);

  }

}
/***********************  function grf_results *****************************/
#define INIT_MAX .01

void grf_results (scan,kal,buf)
int scan;
float kal;
float buf[];
{
  char buffer[80];
  int i, x, y, numpts, first_x, last_x, xscale;
  float yscale;
  static float max=INIT_MAX;

  /* set the max value */
  if ((fabs((double)kal)) > max)
    max = (float) (fabs((double)kal));

  if (scan < DELAY_LENGTH)
    {
    numpts = scan;
    first_x = 0;
    last_x = DELAY_LENGTH-1;
```

```
    }
  else
    {
    numpts = DELAY_LENGTH;
    first_x = scan - (DELAY_LENGTH-1);
    last_x = scan;
    }

  /* Calculate the scaling factor */
  yscale = 150.0/max;
  xscale = 512/DELAY_LENGTH;

  _moveto (0,0);   /* Print the zero axis */
  _lineto (512,0);
  _moveto (0,150);   /* Print the Y axis */
  _lineto (0,-150);

  for(i = 0; i <= 512; i += 64)  /*Print the X axis tick marks */
    {
    _moveto(i, 5);
    _lineto(i, 0);
    }

  for(i = 150; i >= -150; i -= 150)   /* Print the Y axis tick marks */
    {
    _moveto(-4, i);
    _lineto(0, i);
    }

  /* label the axis */
  sprintf(buffer,"%04d", first_x);
  _settextposition(24,7);
  _outtext(buffer);

   sprintf(buffer,"%04d", last_x);
  _settextposition(24,70);
  _outtext(buffer);

  sprintf(buffer,"% 5.4f", max);
  _settextposition(3,0);
  _outtext(buffer);

  sprintf(buffer,"% 5.4f", 0.0);
  _settextposition(13,0);
  _outtext(buffer);

  sprintf(buffer,"% 5.4f", -max);
  _settextposition(23,0);
  _outtext(buffer);

  sprintf(buffer,"SCAN: %5d             : % 7.5f", scan, kal);
  for (i =0; i < 10; i++)
```

```
    buffer[i+13]=hdmsg1[i];
  _settextposition(1,27);
  _outtext(buffer);

  sprintf(buffer,"End key to exit");
  _settextposition(24,35);
  _outtext(buffer);

  /* plot the data */
  _moveto (0, (int) -(buf[0] * yscale));
  for (i=1; i < numpts; i++)
    {
     x = i * xscale;
     y = (int) -(buf[i] * yscale);
     _lineto (x,y);
    }

}
/*************************** function fburst *************************/
int fburst(buffer)
float buffer[];
{

/*  int index, bloc;
  double bval;

  bloc = 0;
  bval = (double) buffer[0];

  for (index = 1; index < INTF_LENGTH; index++)
    if (fabs((double) buffer[index]) > bval)
      {
      bval = fabs((double) buffer[index]);
      bloc = index;
      }

  return (bloc);  */

  int i, max_loc, min_loc;
  float max_val=0.0, min_val=0.0;

  for (i=0; i<INTF_LENGTH; i++)
    if (buffer[i] > max_val)
      {
      max_val = buffer[i];
      max_loc = i;
      }
    else if (buffer[i] < min_val)
      {
      min_val = buffer[i];
      min_loc = i;
      }
```

```
  if (fabs((double) min_val) > max_val)
    return(min_loc);
  else
    return(max_loc);

}
/*************************** function deriv ******************************/
void deriv(buf1, buf2)
float buf1[], buf2[];
{

  int i2n,in,ib,i2b;
  int index, isrt, ifin, ncent;
  float denom;

  /* use the forward difference for the first two points */
  denom = 2.0;
  i2n = 2;
  in = 1;
  for (index=0; index < 2; index++, i2n++, in++)
    buf1[index] = (-buf2[i2n] + 4.0*buf2[in] - 3.0*buf2[index])/denom;


  /* use the backward difference for the last two points */
  i2b = INTF_LENGTH - 4;
  ib = INTF_LENGTH - 3;
  isrt = INTF_LENGTH - 2;
  for (index=isrt; index < INTF_LENGTH; index++, i2b++, ib++)
    buf1[index] = (buf2[i2b] - 4.0*buf2[ib] +3.0*buf2[index])/denom;


  /* use the central difference for the middle points */
  ncent = INTF_LENGTH - 5;
  isrt = 2;
  ifin = INTF_LENGTH - 2;
  i2b = 0;
  ib = 1;
  in = 3;
  i2n = 4;
  denom = 12.0;
  for (index=isrt; index < ifin; index++, i2n++, in++, ib++, i2b++)
     buf1[index] = (buf2[i2b] - 8.0*buf2[ib] + 8.0*buf2[in] -
buf2[i2n])/denom;


}
/****************************** function normal ******************************/
void normal(buffer)
float buffer[];
{

  int index;
```

```
  float ssq = 0.0;

  for (index=0; index < INTF_LENGTH; index++)
    ssq += buffer[index] * buffer[index];

  if (ssq > 0.0)
    ssq = INTF_LENGTH / sqrt(ssq);
  else
    ssq = 1.0;

  for (index=0; index < INTF_LENGTH; index++)
    buffer[index] *= ssq;

}
/************************** function filter **********************************/
void filter(in_buf, out_buf, burst)
float in_buf[];
float out_buf[];
int burst;
{

  int i, j, k;

  for (i=0, k=SEG_OFFSET1+burst-1; i<SEG_LENGTH1; i++, k++)
    {
    out_buf[i] = flt_intercepts1[i];
    for (j=0; j < flt_length1[i]; j++)
      out_buf[i] += flt_coefs1[i][j] * in_buf[ k+flt_offsets1[i][j] ];
    }
}
/************************** function plinear ********************************/
float plinear(in_buf)
float in_buf[];
{

  float dsc_max=-100.0;
  float dsc;
  int i, j, k;

  for (i=0; i < DSC_PASS1; i++)
    {
    dsc = dsc_intercepts1[i];
    for (j=0; j < SEG_LENGTH1; j++)
      dsc += in_buf[j] * dsc_coefs1[i][j];

    if (dsc > dsc_max)
      dsc_max = dsc;
    }

  return(dsc_max);
}
/************************** function kalman **********************************/
```

```
float kalman(scan_num, in_value)
int scan_num;
float in_value;
{
  static float sum=0.0;
  static float sumsq=0.0;
  static float q=0.0;
  static float clkip=0.0;
  static float sigcl2, skip;
  static float prev_input[2*KAL_WIN+1];
  static float beta[2*KAL_WIN+1];
  int nm, i, j;
  float temp, sbase, skm, kal_gain, kal_result, clcov, clkm;

  nm = 2 * KAL_WIN + 1;

  if (scan_num == 0)
    {
    /* setup info for the kalman */
    temp = 3.0/(float)(KAL_WIN*(KAL_WIN+1)*(2*KAL_WIN+1));
    for (i=-KAL_WIN, j=0; i<KAL_WIN+1; i++, j++)
      beta[j] = temp * (float) i;
    }

  if (scan_num < KAL_SETUP)
    {
    sum += in_value;
    sumsq += in_value * in_value;
    return(0.0);
    }

  else if (scan_num == KAL_SETUP)
    {
    sbase = (float) KAL_SETUP;
    sigcl2 = (sbase * sumsq - sum * sum)/(sbase*(sbase-1.0));
    skip = sigcl2;
    return(0.0);
    }

  else
    {
    clkm = clkip;
    skm = skip + q;

    /* compute the kalman gain */
    kal_gain = skm / (skm + sigcl2);

    /* update the intensity covariance */
    clcov = (1.0 - kal_gain) * skm;

    /* update the intensity estimate */
    kal_result = clkm + kal_gain * (in_value - clkm);
```

```
    /* update array of previous values */
    for (i=0; i<nm-1; i++)
      prev_input[i] = prev_input[i+1];
    prev_input[nm-1] = in_value;

    /* update the Q estimate and compute the moving average */
    for (i=0, sum=0.0; i<nm; i++)
      sum += beta[i] * prev_input[i];

    q = sum * sum;
    clkip = kal_result;
    skip = clcov;
    return(kal_result);
    }
}
/************************ function lets_see_it ***************************/
void lets_see_it(device, label, buffer, length)
FILE *device;
char label[];
float buffer[];
int length;
{
  int i;

  if (device == stdout)         /* Output to display */
    {
    fprintf(device,"\n\n\n\n");
    for (i=0; i < length; i += 2)
    fprintf(device, "%s# %4d = %12.3f      %s# %4d = %12.3f\n",
            label, i+1, buffer[i], label, i+2, buffer[i+1]);
    }
  else                          /* Output to the printer */
    {
    fprintf(device,"\r\n\r\n\r\n\r\n");
    for (i=0; i < length; i += 4)
      {
      fprintf(device, "%s# %4d = %12.3f      %s# %4d = %12.3f    ",
              label, i+1, buffer[i], label, i+2, buffer[i+1]);
      fprintf(device, "%s# %4d = %12.3f      %s# %4d = %12.3f\r\n",
              label, i+3, buffer[i+2], label, i+4, buffer[i+3]);
      }
    }
}
/************************ function time_stamp ****************************/
long time_stamp()
  {
  union REGS regs;              /*SETUP FOR REGISTER USE*/
  long tc;

  regs.h.ah = 0;                /*SET ACC FOR TIME TYPE INTERRUPT*/
  int86( 0x1a, &regs, &regs );  /*GENERATE INTERRUPT FOR TIME*/
  tc = (((long) regs.x.cx) << 16) + regs.x.dx;
  return(tc);                   /*RETURN CLOCK TICK*/
  }
```